Continuous-valued Probabilistic Neural Computation in VLSI

Hsin Chen

A thesis submitted for the degree of Doctor of Philosophy.
The University of Edinburgh.
July 19, 2004

Abstract

As interests in implantable devices and bio-electrical systems grow, intelligent embedded systems become important for extracting useful information from continuous-valued, noisy and drifting biomedical signals at sensory interfaces. Probabilistic generative models utilise stochasticity to represent the natural variability of real data, and therefore suggest a potential approach to this application. However, few probabilistic models are amenable to VLSI implementation. This thesis explores the feasibility of realising continuous-valued probabilistic behaviour in VLSI, which may subsequently underpin an intelligent embedded system. In this research, a probabilistic generative model that can model continuous data, with a simple and hardware-amenable training algorithm, has been developed. Based on stochastic computing units with Gaussian noise inputs, this model can adapt its "internal noise" to represent the variability (external noise) of real data. The training algorithm requires only one step of Gibbs sampling and is thus computationally inexpensive in both software and hardware. The capabilities of the model are demonstrated and explored with both artificial and real data. By translating this probabilistic generative model into VLSI implementation, a VLSI system with continuous-valued probabilistic behaviour and on-chip adaptability is further implemented. This not only demonstrates the feasibility of realising continuous-valued probabilistic behaviour in VLSI, but also provides a platform for studying the utility and on-chip adaptability of continuous-valued probabilistic behaviour in VLSI. The system's ability both to model and to regenerate continuous data distributions are explored. As the probabilistic behaviour is introduced by artificially-generated noise, this VLSI system demonstrates computation with noise-induced, continuous-valued probabilistic behaviour in VLSI, and points towards a potential candidate for an intelligent embedded system.

Acknowledgements

The completion of this PhD research has relied very much on my dearest supervisors, colleagues, friends, and family. Without them, I would not have achieved so far and enjoyed this research so much. So I would like to express my greatest thanks to:

my supervisors, *Prof. Alan F. Murray* and *Dr. Martin H. Reekie*, for their inspiring ideas and continual encouragements that lead me through all challenges in this research; in particular, *Prof. Alan F. Murray* for his valuable friendship and supports in every respect.

Dr. Chris William, *Dr. Yee-Whye Teh*, *Dr. Javier R. Movellan*, and *Dr. David Barber* for their patience and knowledgeable discussions that guide me into the field of probabilistic modelling.

all Neural people, particularly *Patrice Fleury*, *Tong-Boon Tang* and *Adria Bofill*, for inspiring discussions and interactions, and for their precious friendship which makes research in Neural group pleasant.

all staff in the School of Engineering and Electronics for all facilities and technical supports; especially the IT team for their efficient and remarkable computing service.

all my friends in Edinburgh for their precious friendship and loads of laughs, which make life in Edinburgh enjoyable and homesick-free; in particular, my girlfriend, *Yu-Ju Chou*, for her continual supports and encouragements, and for sharing every bit of my happiness and bitterness.

the *Committee of Vice-Chancellors and Principals (CVCP)* and the *School of Engineering and Electronics* for the financial supports for this research.

Finally, I would like to express my deepest thanks to my parents, *Jen-I Chen* and *Hsiu-Peng Hsu*, for always providing me with warm and worry-free family supports, which allow me to concentrate on and complete this PhD study.

Contents

List of figures

List of tables

Acronyms and abbreviations

BM = Boltzmann Machine CMOS = Complementary Metal-Oxide-Silicon

CNN = Cellular Neural Networks

CRBM = Continuous Restricted Boltzmann Machine

DAQ = Data-aquisition

DN = Diffusion Network

ECG = Electrocardiogram

HN = Hopfield Network

I-V = Current-to-voltage

LFSR = Linear-Feedback-Shift-Register

MCD = Minimizing Contrastive Divergence

MLP = Multi-Layer-Perceptron

MOS = Metal Oxide Semiconductor

PoE = Products of Experts

PoE/RBM = Product of Experts in the Restricted-Boltzmann-Machine form

RBM = Restricted Boltzmann Machine

RBMrate = Rate-coded Restricted Boltzmann Machine

RNN = Recurrent Neural Network

SVM = Support Vector Machine

VEB = Ventricular Ectopic Beats

VLSI = Very Large Scale Integration

Chapter 1
Introduction

Artificial neural computation refers to brain-inspired algorithms based on simple computing units in parallel and massive connection. The computing unit is called a *neuron* by analogy with neurons in the brain. Probabilistic neural computation refers to the artificial neural computation that employs stochastic neurons, whose inputs merely decide the *probabilities* of outputs, rather than the deterministic values of outputs. Note that the word *deterministic*, contrary to the word *probabilistic*, means the lack of probabilistic behaviour, as will be used in the thesis. This thesis investigates the suggestion that continuous-valued probabilistic behaviour, based on probabilistic neural computation, can be realised effectively in Very Large Scale Integration (VLSI) circuits. Such continuous-valued probabilistic behaviour in VLSI may subsequently underpin an intelligent embedded system. The motivation of this research is described in Sec.1.1, and the contribution to knowledge is clarified in Sec.1.2. Finally, Sec.1.3 outlines the structure of this thesis.

1.1 Motivation

As interests in implantable devices and bio-electrical systems grow [1–7], exposing electronic circuits and sensors to noisy environments becomes unavoidable in many biomedical applications. For example, the works in [1, 2] develop a swallowable pill that contains a variety of sensors and a wireless transmitter, in order to monitor chemical conditions in a patient's gut system. Although a variety of intelligent signal-processing techniques is able to deal with noisy and drifting signals in a digital computer, transmitting all measured raw data out of an

implantable device is very power-consuming. An intelligent embedded system would thus be important to extract useful information at the sensory or bio-electrical interfaces of implantable devices. Such that interfering sources could be reduced and power could be saved by transmitting only useful information. The intelligence here means the ability to classify or to detect novelty in noisy and drifting signals.

In addition to environmental interferences and sensory drifts, the intrinsic noise of electronic devices normally prevents analogue circuits from carrying out precise computation. The intrinsic noise is expected to grow as integrated circuits are fabricated in deep-sub-micron silicon. The immunity against intrinsically-induced computational errors is thus also critical for an intelligent system.

Probabilistic neural computation offers flexible data modelling, wherein the stochasticity both represents the natural variability of real data, and drives a search over solution space during training. Probabilistic neural computation could thus be a potential candidate for an intelligent embedded system to preprocess noisy and drifting signals. Furthermore, probabilistic behaviour may enhance a system's tolerance towards local computational errors, because the probabilistic input-output relationship of computing units could retard the propagation of local computational errors effectively. The VLSI implementation of probabilistic models are therefore of great interests. As most biomedical signals are continuous-valued in nature, probabilistic models capable of modelling continuous data are especially attractive.

However, probabilistic models are seldom both useful in modelling continuous-valued data and amenable to VLSI implementation. Although precise calculations of conditional probability and subsequently reliable Bayesian rules have been demonstrated in VLSI in [8–11], precise calculations are vulnerable to process variations, intrinsic electronic noise, as well as environmental interferences. Such vulnerability at least requires more complicated circuits to overcome

it. Under this concern, VLSI circuits with real probabilistic behaviour, i.e. VLSI circuits with probabilistic input-output relationship, compare favourably with the VLSI implementation of precise calculations.

Some works have contributed to the VLSI implementation of stochastic arithmetic computation [12–18], by coding continuous values as the probability of digital pulses sequences. Computing units communicate with each other by transmitting probabilistic pulses, and the input-pulse streams of a computing unit merely decide the probability for the unit to output a pulse. Such purely-digital and probabilistic computation yields reliable computation and long transmitting distance of continuous values. However, the probabilistic-pulse scheme encourages fully-digital VLSI implementation, in order not only to simplify circuit design but also to be immune to the substrate noise caused by distributed digital pulses. As fully-digital implementation is power- and area consuming, the probabilistic-pulse scheme is not ideal for an intelligent embedded system, in which power and area are expensive.

The VLSI implementation in [19,20] suggests an alternative realisation of probabilistic behaviour in VLSI. Alspector utilised artificially-generated noise to implement the Boltzmann Machine [19], a probabilistic model consisting of *binary stochastic* neurons. The phrase *binary stochastic* means that the neuron's output is binary-valued and follows a probabilistic law. The binary stochastic neurons are implemented by incorporating Gaussian noise at neurons' inputs. The noise inputs cause the neurons to have continuous-valued, probabilistic outputs. The neurons' outputs are then sampled by comparators, for obtaining the binary states of the Boltzmann Machine. The real-world application of the Boltzmann Machine, however, is little explored. This may be attributed to the full and thus complex connections in the Boltzmann Machine, and to the time-consuming training process required by the Boltzmann Machine. Besides, the Boltzmann Machine is a binary-valued model, and is thus not ideally suitable for modelling continuous-valued biomedical signals. Nevertheless, Alspector's works suggests an

interesting and efficient way of introducing probabilistic behaviour in VLSI [21]. The algorithms relating to the Boltzmann Machine also continue to develop, showing a potential way to achieving continuous-valued probabilistic computation in VLSI.

Smolensky proposed the Restricted Boltzmann Machine [22] that utilises a simplified architecture while retaining comparable modelling ability to the Boltzmann Machine. Hinton subsequently proposed the Product of Experts [23, 24], and shows that the Product of Experts in a form of Restricted Boltzmann Machine (the PoE/RBM) is capable of modelling continuous-valued data, with a simple and efficient training rule. Murray further suggested that the PoE/RBM is amenable to hardware implementation [25, 26]. The PoE/RBM has therefore great potential in an intelligent embedded system. However, it is of concern that the PoE/RBM actually employs binary stochastic neurons, which are likely to limit the model's ability to model naturally continuous-valued biomedical signals. More simulations are thus necessary to examine the modelling capability of the PoE/RBM.

To explore continuous-valued probabilistic dynamics, Movellan proposed the Diffusion Network [27,28] composed of *continuous stochastic* neurons. The phrase *continuous stochastic* means that the neuron's output is continuous-valued and follows a probabilistic law. The neurons of the Diffusion Network incorporate Gaussian noise at their inputs, as do the VLSI neurons implemented by Alspector in [19]. The main difference is that the continuous-valued outputs of neurons directly represent the states of a Diffusion Network, while Alspector takes the sign of the continuous-valued outputs to obtain the binary states of a Boltzmann Machine. Movellan also proved that the Diffusion Network can be viewed as a continuous-valued Boltzmann Machine when the stochastic equilibrium of the Diffusion Network is of interest [27]. The theory of the Diffusion Network, together with the VLSI implementation of the Boltzmann Machine, suggest that continuous-valued probabilistic behaviour should be realisable in VLSI and could be underpinned by the Diffusion Network or a development of it.

The developments following on the Boltzmann Machine therefore suggest two approaches to realising continuous-valued probabilistic computation in VLSI. The first is the PoE/RBM which is both able to model continuous data and amenable to hardware implementation. However, more simulations are essential for examining the modelling capability of the PoE/RBM. The other approach is the Boltzmann Machine in VLSI which suggests an efficient way to introduce probabilistic behaviour in VLSI, while continuous-valued probabilistic behaviour induced by this approach remains seldom explored. It is encouraging that the two lines of works could converge to a solution for the principal interest of this research, realising continuous-valued probabilistic neural computation in VLSI

1.2 Contribution to knowledge

As motivated by the works described in Sec.1.1, this project sets out to examine the suggestion:

Continuous-valued probabilistic behaviour can be realised effectively in VLSI circuits.

Such continuous-valued probabilistic behaviour may subsequently underpin an intelligent embedded system. This will be examined in the context of a biomedical application.

With probabilistic models relating to the Boltzmann Machine as a start point, the objectives in this project can be defined as follows:

1. To examine the capability of the PoE/RBM to model continuous data, and to identify the limitations and possible modifications of the PoE/RBM.

2. To identify a continuous-valued probabilistic model that is not only useful in modelling biomedical data, but also amenable to VLSI implementation.

3. To demonstrate all component circuits of the identified model in VLSI, especially for VLSI circuits having continuous-valued probabilistic behaviour.

4. To implement a full system of the identified model in VLSI, in order to study the utility and the on-chip adaptability of continuous-valued probabilistic behaviour in VLSI.

1.3 Chapter layout

The chapters in this thesis can be separated into two parts. The first part, chapter 3 to chapter 4, presents the development of a continuous-valued probabilistic model. Chapter 5 to chapter 8, as the thesis' second part, presents the VLSI implementation of the identified model, from component circuits to a full system on chip. Following the introduction, the contents of these chapters are summarised as follows.

- Chapter 2 reviews the probabilistic models relating to the Boltzmann Machine, and introduces the VLSI implementation of related probabilistic models.
- Chapter 3 discusses the modelling capability of the PoE/RBM.
- Chapter 4 describes the continuous-valued probabilistic model developed in this research, and demonstrate the model's ability to classify both artificial and real biomedical data.
- Chapter 5 derives the modular diagram and digital-control scheme for implementing the full model in VLSI.
- Chapter 6 details the VLSI implementation of a continuous-valued stochastic neuron.
- Chapter 7 addresses the VLSI implementation of the training algorithm for the model.
- Chapter 8 presents the VLSI system of the full probabilistic generative model, and explores the probabilistic behaviour and the on-chip adaptability of this system.
- Chapter 9 concludes the contribution and the future work of this research.

Chapter 2
Literature Review

This chapter contains two parts. The first part, from Sec.2.1 to Sec.2.7, reviews the main features and limitations of probabilistic models relating to the Boltzmann Machine. This brings out the motivation for developing a continuous-valued probabilistic model, and provides essential background for describing the model developed in Ch.4. The second part, from Sec.2.8 to Sec.2.10, reviews critical hardware techniques and considerations for implementing a probabilistic model, particularly the Boltzmann Machine, in VLSI. These techniques provide the basis for implementing the developed model in VLSI. Finally, Sec.2.10 emphasises the distinctive, continuous-valued, probabilistic behaviour this research aims to explore in VLSI.

2.1 Recurrent Neural Network

As probabilistic models relating to the Boltzmann Machine all belong to the family of the *Recurrent Neural Network*, this section first describes the basic concepts and features of the Recurrent Neural Network. Fig.2.1(a) depicts the diagram of a neuron in its general form. The input variables, $\mathbf{s} = \{s_1, s_2, ..., s_j\}$, are weighted by the neuron's *weight* parameters $\{w_{ij}\}$. The function $\mathbf{F}(\cdot)$ then defines the functional relationship between the input $\mathbf{s}$ and the output s_i. For a *deterministic* neuron, $\mathbf{F}(\mathbf{s})$ maps each input $\mathbf{s}$ deterministically to a particular output s_i, while for a *stochastic* neuron, $\mathbf{F}(\mathbf{s})$ is a probability density function that specifies the conditional probability of the output given $\mathbf{s}$, i.e. $Pr(s_i \mid \mathbf{s})$.

Neurons in a fully recurrent neural network are connected as shown in Fig.2.1(b). The output of each neuron is fed back to the inputs of all neurons, and is therefore dependent on the

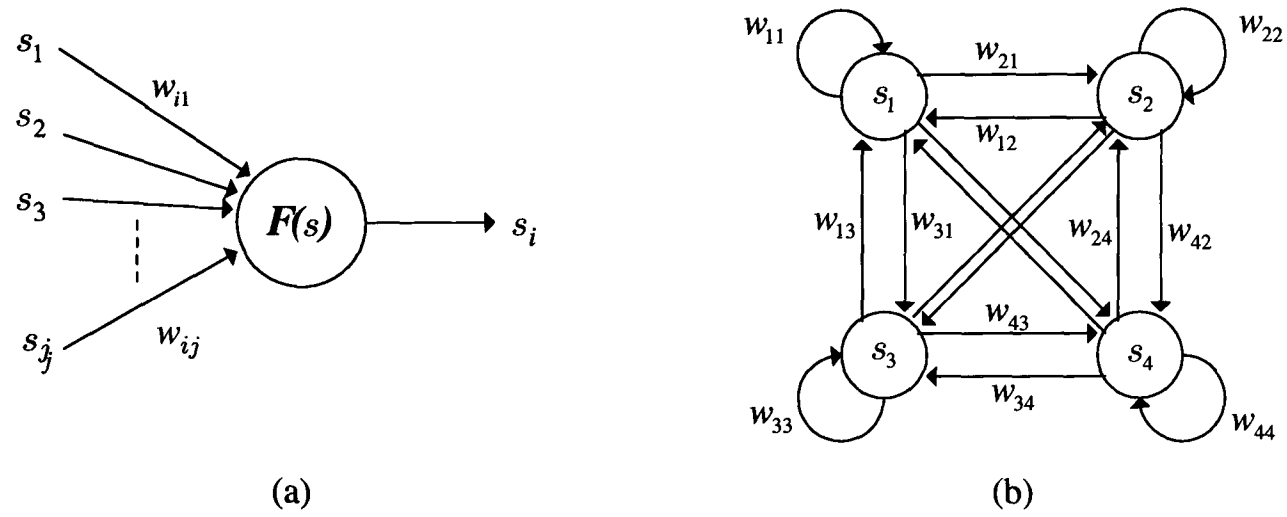


Figure 2.1: *The diagrams of (a)a single neuron (b) a recurrent neural network with four neurons*

output of all neurons. Such dependence prevents neurons from being updated simultaneously in software simulation. Only one neuron is selected and updated at each time step n, and the selection process is purely random such that all neurons have equal opportunity to be selected. For stochastic neurons, updating a neuron means sampling one value from the neuron's output probability density, $Pr(s_i \mid \mathbf{s})$, and the sampling process is called *Gibbs sampling*. The output of a neuron i can thus be written as a function of time, $s_i[n]$.

Let the state of a recurrent neural network be the vector including the outputs of all neurons, $\mathbf{s} = \{s_i\}$. The *dynamics* of a recurrent neural network refer to the evolution of $\mathbf{s}$ over time, given an initial state $\mathbf{s}[0]$. This leads to the general consideration about the existence of *equilibrium*. For a *deterministic* recurrent neural network [1], the *equilibrium* refers to the convergence of $\mathbf{s}$ to a final equilibrium state $\mathbf{s}_\infty$ when $t \rightarrow \infty$. Note that a deterministic recurrent neural network can have more than one equilibrium state. For a *probabilistic* recurrent neural network, *equilibrium* means that the probability density of $\mathbf{s}$ converges to a fixed density over the state space when $t \rightarrow \infty$. Therefore, if the neurons in Fig.2.1(b) are deterministic, the outputs of all neurons will become constant after a sufficiently long updating process, provided that the

[1] a recurrent neural network is called *deterministic* if it employs *deterministic* neurons, while called *probabilistic* if it employs *stochastic* neurons

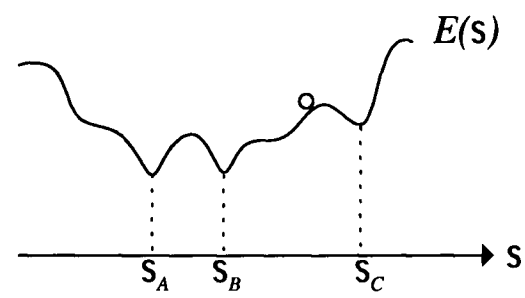


Figure 2.2: *An example of an energy landscape in a one-dimensional state space*

equilibrium of the network exists. If the neurons are stochastic, the probability that the network is in a particular state $\mathbf{s}_\alpha$ will become constant after sufficient Gibbs sampling, despite the fact that the network's actual state $\mathbf{s}$ remains in a state of change.

The behaviour of a recurrent neural network is able to be described by the principles of statistical physics. A recurrent neural network is usually associated with an *energy function* $E(\mathbf{s})$ that assigns to each state $\mathbf{s}$ a real-valued *energy*. Plotting the energy values in the state space of the network forms an *energy landscape* in the state space, which provides a better understanding of the dynamics and the equilibrium of the recurrent neural network. Fig.2.2 shows an example of an energy landscape in a one-dimensional state space. The dynamics of the associated recurrent neural network can be viewed as a ball travelling in the energy landscape. Assume that the equilibrium of the network exists. If the recurrent neural network is *deterministic*, the states corresponding to the minima of the energy function (e.g. $\mathbf{s}_A$ and $\mathbf{s}_B$ in Fig.2.2) are the equilibrium states of the recurrent neural network. This will become clear in Sec.2.2. If the recurrent neural network is *probabilistic*, particularly for the example of the Boltzmann Machine described in Sec.2.3, a state with lower energy will have higher probability when the network reaches equilibrium. For example, if Fig.2.2 represents the energy function of a Boltzmann Machine, the Boltzmann Machine is more probably in the equilibrium state $\mathbf{s}_A$ than in the state $\mathbf{s}_C$.

Simulating a recurrent neural network in software is time-consuming because only one neu-

ron can be updated at each execution step. The hardware implementation of a recurrent neural network, on the contrary, allows all neurons to be updated in parallel, as demonstrated in [29] and will be explained in Sec.2.8. Analogue VLSI is especially attractive, as analogue circuits have been shown to be efficient in simulating the dynamics of a recurrent neural network [30]. Hardware implementation of *probabilistic* recurrent neural networks is thus of great interest. The following sections will describe various recurrent neural networks, from the Hopfield Network with deterministic neurons, to the Boltzmann Machine with binary stochastic neurons, and to the Diffusion Network with continuous stochastic neurons.

2.2 The Hopfield Network

2.2.1 The binary Hopfield Network

The *binary* Hopfield Network refers to the Hopfield Network proposed in [31] that employs *binary deterministic* neurons, as shown in Fig.2.3(a). The input θ_i represents the *threshold* value of the neuron, and s_i represents the output of the neuron. The weighted inputs and the threshold are summed and passed through a *signum* function, yielding a binary-valued output. The deterministic function of the neuron is thus written as

$$s_i = \text{sgn}\left[\sum_j w_{ij}s_j - \theta_i\right] \tag{2.1}$$

The binary Hopfield Network connects neurons in a *symmetric* recurrent structure ($w_{ij} = w_{ji}$) without self-feedback ($w_{ii} = 0$), as shown in Fig.2.3(b). As $w_{ij} = w_{ji}$, each bidirectional arrow in Fig.2.3 corresponds to two uni-directional arrows in Fig.2.1(b). To describe the dynamics of

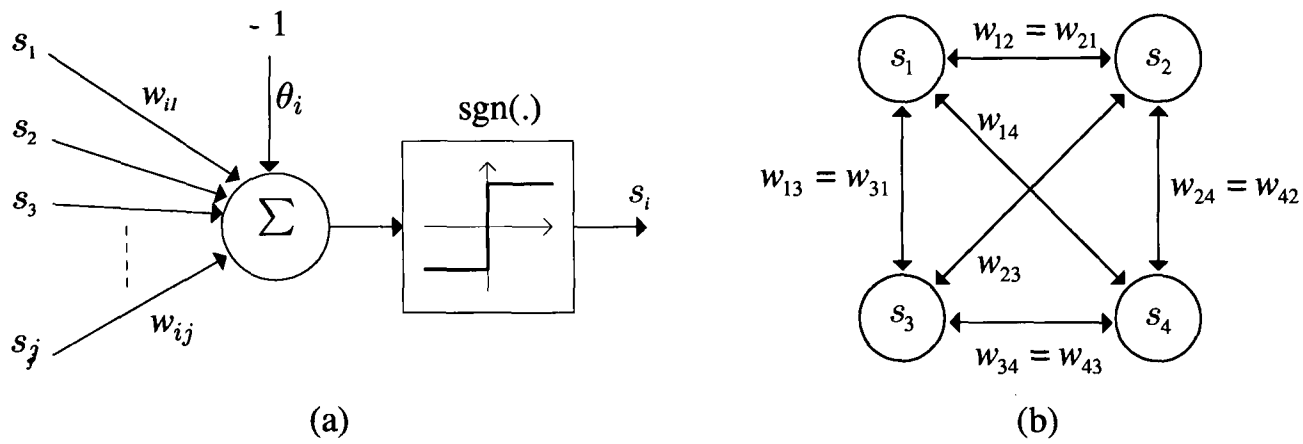


Figure 2.3: *The diagrams of (a)a binary deterministic neuron used in the binary Hopfield Network (b)A Hopfield Network with four neurons*

the binary Hopfield Network, Hopfield defines an energy function as [31]

$$E_{bHN} = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i \tag{2.2}$$

Let neurons be updated according to Eq.(2.1), and Δs_i represent the output change of neuron i. Eq.(2.2) indicates that Δs_i introduces an energy change equal to

$$\Delta E = -\Delta s_i \left(\sum_j w_{ij} s_j - \theta_i \right) \tag{2.3}$$

As Eq.(2.1) indicates that Δs_i and $(\sum_j w_{ij} s_j - \theta_i)$ always have the same sign, the energy change ΔE is always negative. If the dynamics of a binary Hopfield Network is viewed as a ball moving in the energy landscape in Fig.2.2, such a consistently-negative energy change means that the ball "consistently moves downhill". Updating neurons according to Eq.(2.1) thus always leads the binary Hopfield Network to a state corresponding to a global or local energy minimum, i.e. an equilibrium state.

With this feature, a binary Hopfield Network with N neurons is often employed as a *content-addressable memory* [31, 32], which is able to *store* a set of N-dimensional binary

vectors (memories) $\{\mathbf{v}_\alpha\}$ by setting

$$w_{ij} = \frac{1}{N} \sum_{\alpha} \mathbf{v}_{\alpha,i} \mathbf{v}_{\alpha,j} \tag{2.4}$$

where $\mathbf{v}_{\alpha,i}$ denotes the i-th component of vector $\mathbf{v}_\alpha$. Eq.(2.4) stores the vectors $\{\mathbf{v}_\alpha\}$ as the energy minima of the network. Therefore, given an initial state with fully-random values or with incomplete content of a particular $\mathbf{v}_\alpha$, the binary Hopfield Network will converge towards an energy-minimum state, which is normally one of the stored vectors. If the initial state contains an incomplete version of $\mathbf{v}_A$ and the converged equilibrium state is $\mathbf{v}_A$, this particular function is equivalent to retrieving the memorised $\mathbf{v}_A$ from its incomplete content, as suggested by the name *content-addressable memory*.

2.2.2 The continuous Hopfield Network

The capacity of the binary Hopfield Network is limited by the number of neurons [32]. The maximum number of memories an N-neuron binary Hopfield Network can store is $N/(2\ln N)$, in order to ensure the correspondence of $\{\mathbf{v}_\alpha\}$ to equilibrium states. To remove this limitation and to achieve a more biologically-plausible model, Hopfield generalised the binary Hopfield Network into the *continuous* Hopfield Network [33]. The continuous Hopfield Network retains the same recurrent structure as Fig.2.3(b), while employing *continuous deterministic* neurons that incorporate the parameters of membrane capacitance C_i and transmembrane resistance R_i, analogous to biological neurons. Fig.2.4 shows the continuous deterministic neuron as electronic components, where x_i represents a voltage and g_i a controlled current source whose current equals the total input $\sum_j w_{ij} s_j$. The neuron's output function is also transformed to a *sigmoid* function, $\varphi(\cdot)$, which has a monotonically-increasing input-output relationship (Fig.2.4). The neuron thus has a continuous-valued output and changes, in continuous time and

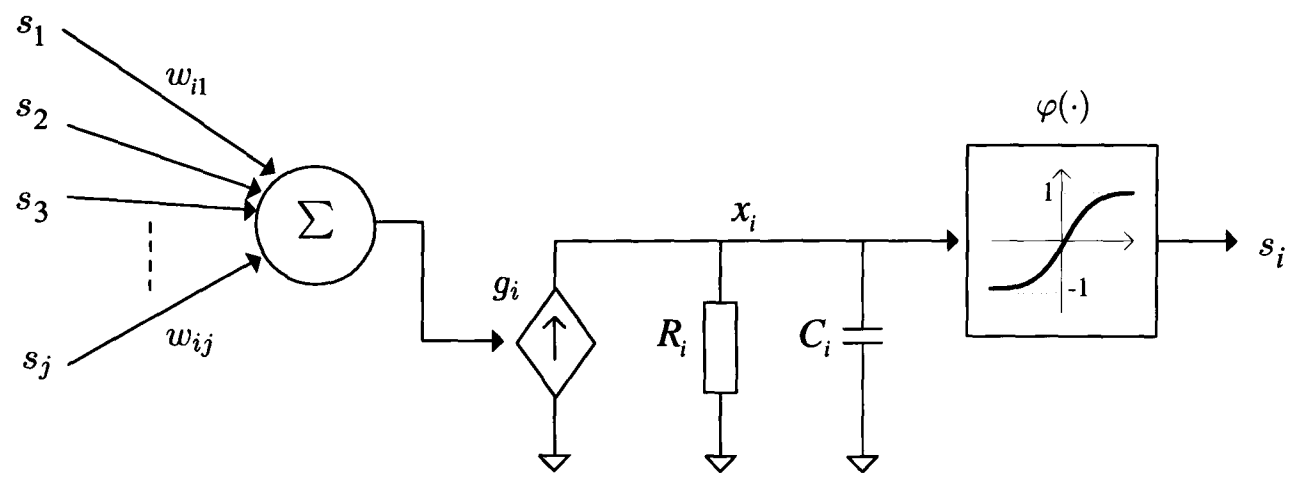


Figure 2.4: *The neuron of the continuous Hopfield Network in terms of electronic components*

deterministically, according to [2].

$$C_i(dx_i(t)/dt) = \sum_j w_{ij}s_j(t) - x_i(t)/R_i, \qquad (2.5)$$

$$\text{where} \quad x_i(t) = \varphi^{-1}(s_i(t))$$

where $s_i(t)$ represents the output of neuron i at time t. The energy function associated with the continuous Hopfield Network is [33]

$$E_{cHN} = -\sum_{i<j} w_{ij}s_i s_j + \sum_i (1/R_i)\int_0^{s_i} \varphi^{-1}(s)ds \qquad (2.6)$$

As in Eq.(2.1), the deterministic dynamics in Eq.(2.5) always lead the continuous Hopfield Network towards an energy minimum, i.e. an equilibrium state [33]. Fig.2.5 shows that the equilibrium states of a continuous Hopfield Network can take any value in its continuous state space, while the equilibrium states of a binary Hopfield Network are always at the corners of its hyper-cubic space. The continuous Hopfield Network thus has a more flexible representational ability than the binary Hopfield Network.

[2]The voltage change of a capacitor C is proportional to the total current I input to the capacitor, i.e. $C\frac{dv}{dt} = I$

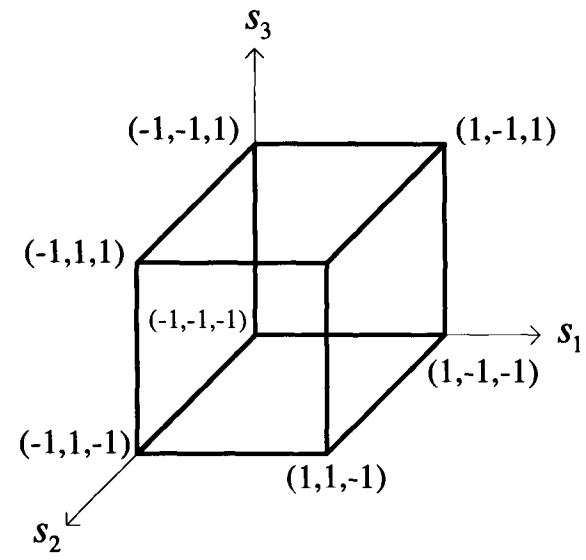


Figure 2.5: *The equilibrium states of a continuous Hopfield Network with three neurons can be any points in the cube, while a binary Hopfield Network with three neurons can only have equilibrium states at the eight corners of the cube*

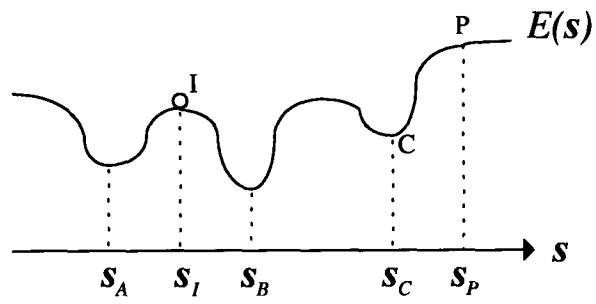


Figure 2.6: *The energy landscape that illustrates the limitation of the Hopfield Network*

2.2.3 Discussion

The Hopfield Network, however, still suffers from several limitations. When the Hopfield Network functions as a *content-addressable memory*, a retrieved equilibrium state does not necessarily relate to the incomplete content of its corresponding initial state. Imagine a one-dimensional energy landscape with several local minima, as illustrated by Fig.2.6. Let the initial state be $\mathbf{s}_I$. As the state (ball) always moves downhill, the final equilibrium state can be either $\mathbf{s}_A$ or $\mathbf{s}_B$, no matter which is the real completion of $\mathbf{s}_I$. In addition, the equilibrium states of a Hopfield Network sometimes include *spurious states* that do not correspond to any pattern stored by Eq.(2.4) [34]. For example, $\mathbf{s}_C$ in Fig.2.6 is an equilibrium state, but $\mathbf{s}_C$ does not

necessarily correspond to any of the stored vectors $\{\mathbf{v}_\alpha\}$. If the initial state is $\mathbf{s}_P$, the Hopfield Network will be trapped in $\mathbf{s}_C$, a *spurious* equilibrium state. These limitations are addressed by evolving the deterministic Hopfield Network into the probabilistic model called the Boltzmann Machine [35], as described in the next section.

2.3 The Boltzmann Machine

2.3.1 From deterministic to probabilistic

The Boltzmann Machine proposed by Hinton [35] can be viewed as a stochastic transformation of the binary Hopfield Network. It differs from the binary Hopfield Network as follows.

- The neurons of the Boltzmann Machine have *binary stochastic behaviour.*
- The use of *hidden neurons.*
- The *simulated annealing algorithm* [36] is applicable to the Boltzmann Machine to find an optimum solution for tasks such as incomplete-pattern completion.

Fig.2.7(a) shows the *binary stochastic* neuron employed by the Boltzmann Machine. Unlike the binary neuron in the Hopfield Network (Fig.2.3(a)), the total input of the neuron decides only an output probability, $\sigma(x_i)$, as given in the following equation. A binary-valued output, either 0 or 1 [3], is then sampled according to this probability.

$$Pr(s_i = 1 \mid x_i) = \sigma(x_i) = \frac{1}{1 + \exp(-x_i/T)}, \qquad \text{where} \quad x_i = \sum_j w_{ij} s_j - \theta_i \tag{2.7}$$

[3] A network with binary states of $\{0, 1\}^N$ can always be transformed to a network with binary states of $\{-1, 1\}^N$

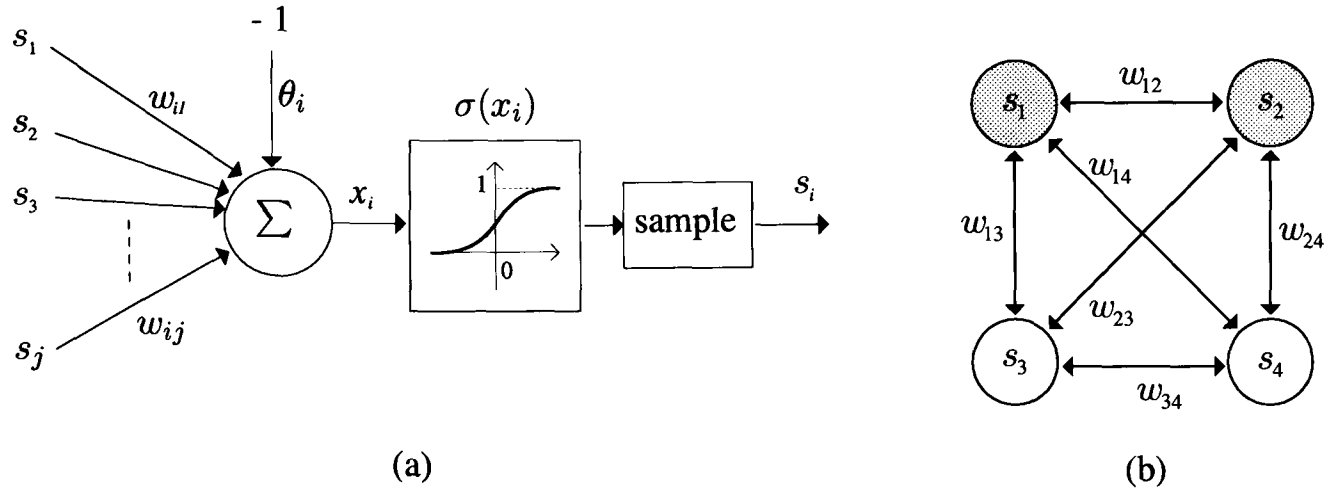


Figure 2.7: *(a)The binary stochastic neuron in the Boltzmann Machine (b)A Boltzmann Machine with two visible(white-coloured) and two hidden(grey-coloured) neurons*

$\sigma(\cdot)$ represents the *logistic* function, a sigmoid function whose output ranges from 0 to 1 (Fig.2.7(a)), and T is referred to as the *temperature*, controlling the slope of the sigmoid function.

The Boltzmann Machine connects neurons in a symmetric network without self-feedback, and the neurons of the Boltzmann Machine are divided into *visible* and *hidden* neurons. Fig.2.7(b) illustrates a Boltzmann Machine with two visible (white) and two hidden (grey) neurons. Visible neurons, similar to the neurons of the Hopfield Network, relate directly to the data modelled by the Boltzmann Machine. If the data are N-dimensional vectors, the number of visible neurons is normally set to N, such that each visible neuron corresponds to one dimension of the data. The function of hidden neurons, however, differs from that of visible neurons. Instead of corresponding to the data, hidden neurons provide flexibility for modelling complex interactions between visible neurons, i.e. the complex statistical structure underlying the data. The representational ability of the Boltzmann Machine is therefore inherently better than that of the binary Hopfield Network.

The energy function of the Boltzmann Machine is the same as that of the binary Hopfield Network, and is re-written in the following way:

$$E_{BM} = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i \tag{2.8}$$

The energy function together with the updating equation for neurons (Eq.(2.7)), however, leads to a dynamical behaviour different from that of the Hopfield Network. If the state of neuron i changes from 1 to 0, Eq.(2.8) indicates that the transition will introduce an energy change of $\Delta E_i = \sum_j w_{ij} s_j - \theta_i$ to the Boltzmann Machine. As ΔE_i is equivalent to x_i in Eq.(2.7), Eq.(2.7) can be re-written as

$$Pr(s_i = 1 \mid \Delta E_i) = \frac{1}{1 + \exp(-\Delta E_i / T)} \tag{2.9}$$

Assuming that ΔE_i is negative-valued, Eq.(2.9) reveals that a negative-valued energy change merely yields a higher "probability" for s_i to flip to 0, but does not update s_i to 0 deterministically. Imagine the dynamics of the Boltzmann Machine as a ball moving in the energy landscape in Fig.2.8 [4]. Let $\mathbf{s} = \{s_i\}$ denotes the state of the Boltzmann Machine. Even if the ball moves into a local minimum such as the state $\mathbf{s}_C$, Eq.(2.9) indicates that the Boltzmann Machine can "go uphill occasionally", and avoid being trapped by a local minimum, or a spurious state as in the example of the Hopfield Network.

From another perspective, the probabilistic behaviour prevents the Boltzmann Machine from stabilising into a single state. The Boltzmann Machine has a non-zero probability for any of its states. The equilibrium of the Boltzmann machine therefore refers to a probability distribution over its state space. Gibbs sampling neurons according to Eq.(2.7) leads the Boltz-

[4]Note that the state space of the Boltzmann Machine is actually a hyper-cube space, rather than a continuous space. The representation in Fig.2.8 can be imagined as putting all possible binary states $\{\mathbf{s}_\alpha\}$ one-by-one onto the axis of $\mathbf{s}$

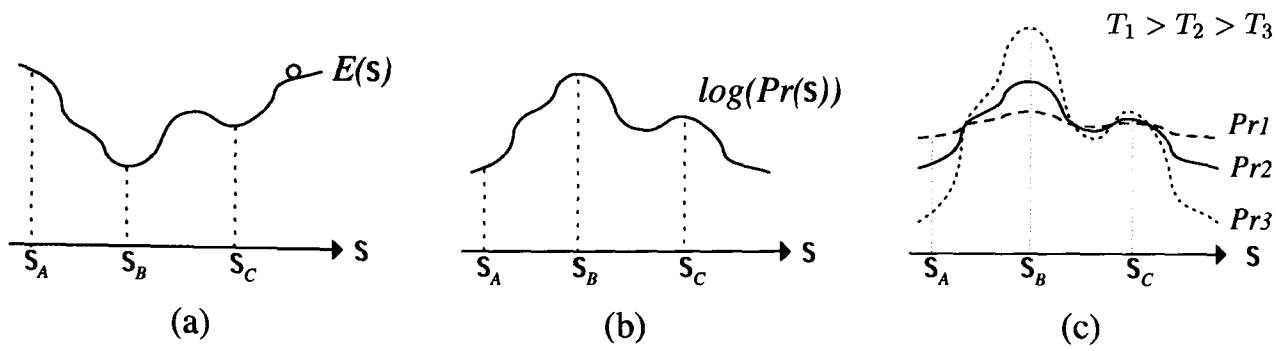


Figure 2.8: *(a)The energy landscape of a Boltzmann Machine (b)The corresponding logarithmic probability density in Boltzmann equilibrium (c)The effect of temperature T on the equilibrium probability distribution*

mann Machine towards a *thermal equilibrium*, called the *Boltzmann equilibrium* [35], in which the Boltzmann Machine is in state s with a constant probability $Pr(\mathbf{s})$ of

$$Pr(\mathbf{s}) = \frac{1}{Z}\exp\left(-\frac{E(\mathbf{s})}{T}\right)$$
$$\text{with} \quad Z = \sum_{\mathbf{s}} \exp\left(-\frac{E(\mathbf{s})}{T}\right) \tag{2.10}$$

where $E(\mathbf{s})$ represents the energy of state s and Z is a normalising factor summing over all possible states. Eq.(2.10) indicates that the probability of being in a particular state is inversely proportional to the energy of the state. For example, state $\mathbf{s}_A$ in Fig.2.8(a) has a higher energy than state $\mathbf{s}_B$. The Boltzmann Machine is thus more likely to be in state $\mathbf{s}_B$ than in state $\mathbf{s}_A$ when it is in equilibrium. More specifically, turning the curve (the energy landscape) in Fig.2.8(a) upside down gives the logarithmic probability density of Boltzmann equilibrium , as shown in Fig.2.8(b), which is proportional to $Pr(\mathbf{s})$ in Eq.(2.10).

Eq.(2.10) further reveals that the *simulated annealing* algorithm [36] can be used with the Boltzmann Machine. Simulated annealing specifies an *annealing schedule* that sets the temperature T initially to a high value T_{ini} and lowers the temperature gradually to a final temperature T_{final}. The effect of temperature change on equilibrium probability distribution is illustrated in Fig.2.8(c). As the temperature becomes lower, from T_1 to T_3, the steepness of the probability

distribution increases, from Pr_1 to Pr_3. Therefore, only the state with minimum energy has nonzero probability when the temperature becomes infinitely small. The initially-high temperature T_{ini} thus causes the Boltzmann Machine to explore the global structure of its state space first, and lowering the temperature gradually allows the Boltzmann Machine to capture more detail. A near-zero temperature T_{final} finally forces the Boltzmann Machine to stabilise into one lowest-energy state, provided that the gradual change is slow enough for the Boltzmann Machine to "relax" to *thermal equilibrium* at each temperature [5].

The simulating annealing scheme is useful for optimisation tasks, in which the energy function is normally treated as a cost function. The lowest-energy state then corresponds to the optimised solution with least cost. In the task of incomplete-pattern completion, for example, the simulated annealing algorithm guarantees that the Boltzmann Machine will find the completion of the initial incomplete content, while the Hopfield Network does not necessarily retrieve a correct completion.

2.3.2 Training the Boltzmann Machine

Since the equilibrium of the Boltzmann Machine refers to an equilibrium probability distribution over its state space, the Boltzmann Machine is called a *generative* model that learns to "regenerate" data points having the same probability distribution as its training data. Let $\mathbf{v}$ denote the subset of state $\mathbf{s}$, including only visible neurons' outputs. Let the probability distribution of training data be represented as $P^0(\mathbf{v})$, and the probability distribution of Boltzmann equilibrium as $P^\infty(\mathbf{v})$. The difference between $P^0(\mathbf{v})$ and $P^\infty(\mathbf{v})$ is measurable by the

[5] To ensure convergence to global minimum, the gradual change of temperature should follow the rule $T_k \geq \frac{T_{ini}}{\log(1+k)}$ [32]

Kullback-Leibler Divergence [35]

$$G = \sum_{\mathbf{v}} P^0(\mathbf{v}) \log \frac{P^0(\mathbf{v})}{P^\infty(\mathbf{v})} \tag{2.11}$$

"Minimising the KL-divergence" above gives the following training rule [35]

$$\begin{aligned} \Delta w_{ij} &\propto -\frac{\partial G}{\partial w_{ij}} \\ &= \eta_w \left(\langle s_i s_j \rangle_0 - \langle s_i s_j \rangle_\infty \right) \end{aligned} \tag{2.12}$$

where η_w defines the learning rate, s_i and s_j represent the states of neuron i and j, respectively. $\langle \cdot \rangle_0$ denotes the expectation value over the training data, and $\langle \cdot \rangle_\infty$ the expectation value over the samples from equilibrium distribution.

Eq.(2.12) reveals that the training process comprises two phases, called the *clamping* and the *free-running* phases [35]. Let $\mathbf{v}_\alpha$ represent a particular training datum. In the *clamping* phase, each training datum $\mathbf{v}_\alpha$ is "clamped" to visible neurons, i.e. fixing the outputs of visible neurons as $\mathbf{v}_\alpha$. Hidden neurons, on the contrary, are Gibbs sampled freely until equilibrium is reached. The first correlation term in Eq.(2.12) is then computed and averaged over the training data. In the *free-running* phase, both visible and hidden neurons are freely Gibbs sampled until equilibrium is reached, and the second correlation term in Eq.(2.12) is then computed. The weights are subsequently updated once, and the two-phase training is repeated until weight changes are negligible. Incorporating the *simulated annealing* algorithm optimises the training process, by lowering temperature gradually [32]. Ideally, Gibbs sampling from the equilibrium distribution of a trained Boltzmann Machine regenerates data points with the same distribution as the training data.

2.3.3 Discussion

The Boltzmann Machine provides a good example of utilising "imprecise" computing units to achieve reliable computation and thus to enrich modelling ability. The time-consuming relaxation for thermal equilibrium and the slow simulated-annealing schedule, however, make the Boltzmann Machine impractical in many real applications. Although Peterson and Anderson proposed the Mean-Field Approximation [37] to speed up relaxation search, this deterministic approximation becomes infeasible when the Boltzmann Machine has many hidden neurons with complex inter-connections [32]. On the contrary, the hardware implementation of the Boltzmann Machine [19, 20, 38, 39] shows that both relaxation search and simulated annealing become computationally-inexpensive in analogue hardware. Although massive connections between neurons remain costly in hardware, the architecture can be further simplified based on the Restricted Boltzmann Machine, as will be described in Sec.2.5. The next section reviews the Diffusion Network [27], which forms a *continuous* Boltzmann Machine under some conditions. The word *continuous* indicates that the neurons of the Diffusion Network have continuous-valued probabilistic behaviour.

2.4 The Diffusion Network

The Diffusion Network proposed by Movellan [27, 28, 40, 41] transforms the continuous Hopfield Network into a stochastic form, and generalises the Boltzmann Machine into a continuous-valued version. The first subsection will describe the transformation from the continuous Hopfield Network to the probabilistic Diffusion Network. The second subsection will then specify the conditions under which the Diffusion Network is a continuous Boltzmann Machine.

2.4.1 From deterministic to probabilistic

The differential equation that governs the *deterministic* dynamics of the continuous Hopfield Network, Eq.(2.5), can be rewritten as

$$dx_i(t) = \mu_i(t) \cdot dt \tag{2.13}$$

where $\mu_i(t)$ is a deterministic *drift* term equal to

$$\mu_i(t) = \frac{1}{C_i}\left(\sum_j w_{ij} s_j(t) - x_i(t)/R_i\right) \tag{2.14}$$

The Diffusion Network includes *probabilistic* dynamics by incorporating an extra term of *Brownian motion* $dB_i(t)$:

$$dx_i(t) = \mu_i(t) \cdot dt + \sigma \cdot dB_i(t) \tag{2.15}$$

The increment $(dB_i(t+dt) - dB_i(t))$ is a Gaussian random variable with zero mean and variance dt. $\sigma \cdot dB_i(t)$ is thus equivalent to Gaussian noise whose variance is scaled by a constant σ. Let $x_i(t)$ denote the state of neuron i in a Diffusion Network. Eq.(2.15) indicates that the continuous-time change of $x_i(t)$ is determined by both the deterministic drift component $\mu_i(t)$ and by the noise component, $\sigma \cdot dB_i(t)$. The dynamics governed by Eq.(2.15) are normally called a *diffusion process*. To visualise the main difference between the neurons of the Diffusion Network and the neurons of continuous Hopfield Network, substituting Eq.(2.14) into Eq.(2.15) gives

$$dx_i(t) = \frac{1}{C_i}\left(\sum_j w_{ij} s_j(t) - \frac{x_i(t)}{R_i}\right) \cdot dt + \sigma \cdot dB_i(t) \tag{2.16}$$

where $s_j(t) = \varphi_j(x_j(t))$ and $\varphi_j(\cdot)$ represents a sigmoid function such as $\tanh(\cdot)$. Fig.2.9 shows the diagram of the neuron defined by Eq.(2.16) as electronic components. Clearly, the

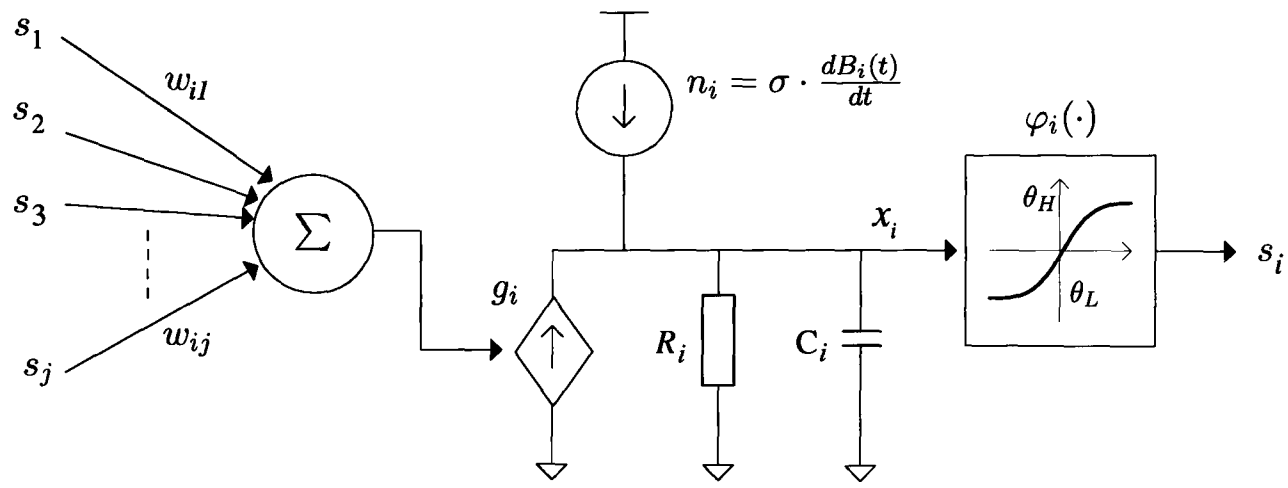


Figure 2.9: *The neuron of the Diffusion Network in terms of electronic components*

neuron in Fig.2.9 differs from the neuron of the Hopfield Network (Fig.2.4) largely by the inclusion of a noise source n_i. The noise source causes both $x_i(t)$ and $s_i(t)$ to be probabilistic, and thus the neuron's behaviour to be "*continuous-valued stochastic*". Given the deterministic drift $\mu_i(t)$ and the constant σ, the conditional probability of $x_i(t)$ is a Gaussian function as in Eq.(2.17), while the conditional probability of $s_i(t)$ depends upon the sigmoid function $\varphi_i(t)$ as Eq.(2.18).

$$\begin{aligned} Pr(\, x_i(t) \mid \mu_i(t), \sigma) &= \mathcal{G}(\mu_i(t), \sigma^2) \\ &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x_i(t) - \mu_i(t))^2}{2\sigma^2}\right) \end{aligned} \tag{2.17}$$

$$Pr(\, s_i(t) \mid \mu_i(t), \sigma) = \frac{\mathcal{G}(\mu_i(t), \sigma^2)}{\varphi_i'(x_i)} \tag{2.18}$$

The Diffusion Network connects neurons in a general recurrent structure, as shown in Fig.2.10(a). The structure contains both asymmetric connections ($w_{ij} \neq w_{ji}$) and self-feedback. By analogy with the Boltzmann Machine, the neurons in Fig.2.10 are also divided into visible and hidden neurons, as indicated by different colours in Fig.2.10(a). This general structure

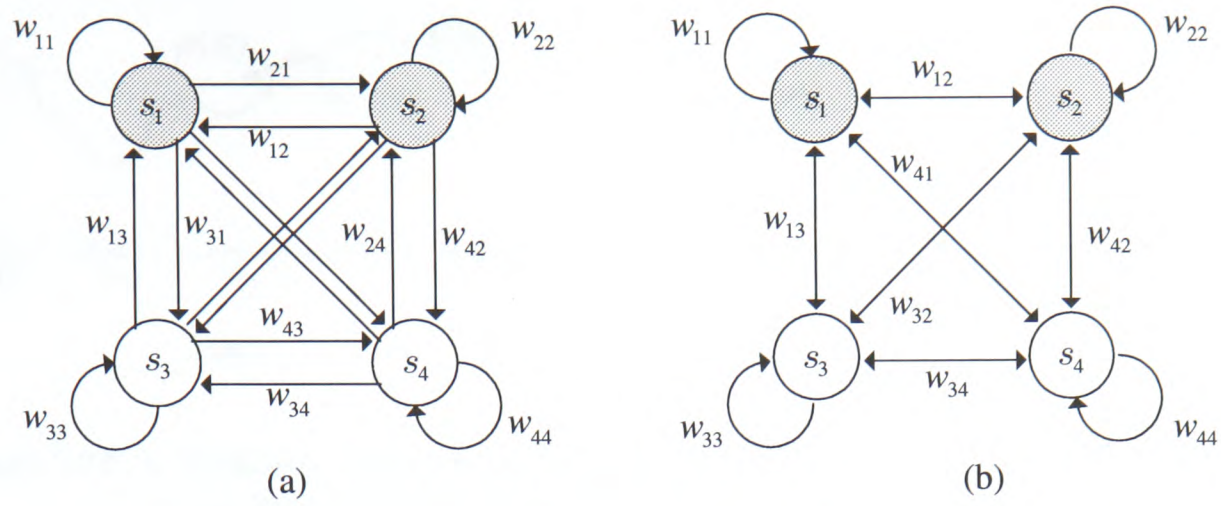


Figure 2.10: *A Diffusion Network of two visible(white-coloured) and two hidden(grey-coloured) neurons with (a) asymmetric connections (b)symmetric connections*

maximises the representational power of the Diffusion Network, but also complicates its dynamics. Let $\mathbf{x}_t = \{x_i(t)\}$ denote the state of a Diffusion Network at time t. The dynamics refer to the evolution of state $\mathbf{x}_t$ in continuous time, governed by Eq.(2.15). Although the equilibrium condition for the Diffusion Network (when $t \to \infty$) is defined in [27], the explicit form of equilibrium is difficult to determine in general. There exists, however, a special case within which the equilibrium form is tractable and is the form of *Boltzmann equilibrium*, as described in the next subsection.

2.4.2 The symmetric Diffusion Network and continuous Boltzmann Machine

Consider a Diffusion Network with symmetric connections ($w_{ij} = w_{ji}$), as shown in Fig.2.10(b). The Diffusion Network is associated with an energy function $E(\mathbf{x}_t)$ that defines an energy landscape over its continuous state space. Let $\mu_t = \{\mu_i(t)\}$ denote the drift vector composed of the drift terms of all neurons. If μ_t satisfies the following condition

$$\mu_t = -\nabla E(\mathbf{x}_t) \tag{2.19}$$

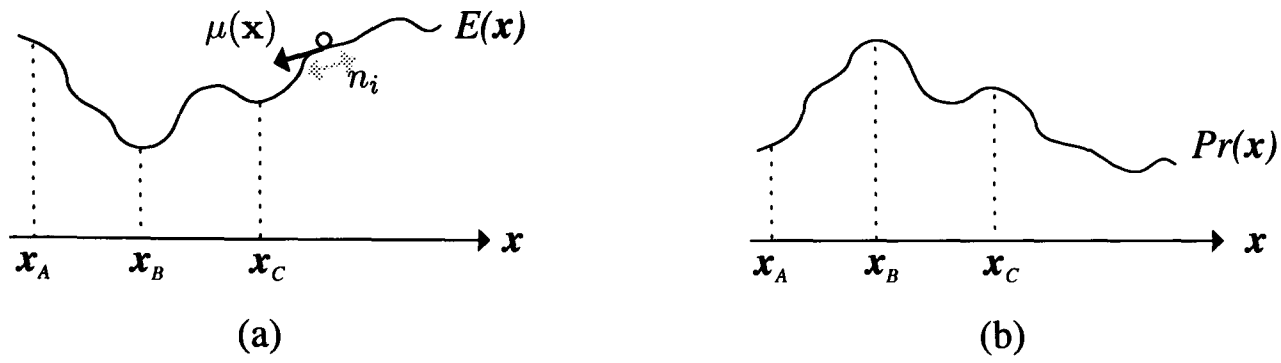


Figure 2.11: *A Diffusion Network of two visible(white-coloured) and two hidden(grey-coloured) neurons with (a) asymmetric connections (b)symmetric connections*

the diffusion process in Eq.(2.15) always leads the Diffusion Network to a well-defined equilibrium [27], in which the Diffusion Network is in state $\mathbf{x}$ with a constant probability $Pr(\mathbf{x})$ of

$$\begin{aligned} Pr(\mathbf{x}) &= \frac{1}{Z}\exp\left(-E(\mathbf{x})\right) \\ \text{with} \quad Z &= \int_{\mathbf{x}} \exp\left(-E(\mathbf{x})\right) d\mathbf{x} \end{aligned} \tag{2.20}$$

Note that the subscript t of $\mathbf{x}_t$ is removed to denote the equilibrium state. Z is a normalising factor to ensure $\int Pr(\mathbf{x})d\mathbf{x} = 1$. Eq.(2.20) and Eq.(2.10) reveal that the Diffusion Network has an equilibrium form similar to that of the Boltzmann Machine, except that the equilibrium is a distribution over the *continuous* space and there is no temperature factor. The Diffusion Network under the condition of Eq.(2.19) can therefore be viewed as a *continuous Boltzmann Machine*. If the energy landscape of the continuous Boltzmann Machine is the one-dimensional curve in Fig.2.11(a), the corresponding probability density of Boltzmann equilibrium is inversely proportional to the energy, as shown in Fig.2.11(b). Fig.2.11(a) also provides a better understanding of the diffusion process governed by Eq.(2.15). Imagine the diffusion process as a ball moving in the energy landscape. Eq.(2.19) indicates that the deterministic drift force always pulls the ball towards downhill, as depicted by the black arrow in Fig.2.11(a). The noise disturbance, on the contrary, introduces a random movement as indicated by the grey arrows,

enabling the ball to "go uphill occasionally". The diffusion process with $\mu = -\nabla E$ therefore "mimics" the dynamics of the Boltzmann Machine in *continuous* space.

2.4.3 Training the symmetric Diffusion Network

The symmetric Diffusion Network discussed in this subsection refers particularly to the Diffusion Network in the form of the Continuous Boltzmann Machine. As the symmetric Diffusion Network has an explicit form of equilibrium, the symmetric Diffusion Network is able to be trained, as a generative model, to model continuous data distributions. Let λ_i refer to any parameter of neuron i in a symmetric Diffusion Network. By minimising the Kullback-Leibler divergence, the training rule for λ_i is [27]

$$\Delta\lambda_i \propto \langle S_{\lambda_i} \rangle_0 - \langle S_{\lambda_i} \rangle_\infty \tag{2.21}$$

where S_{λ_i} is called the *system covariate*, and is defined as the negative derivative of energy function,i.e. $S_{\lambda_i} = -\partial E / \partial \lambda_i$ [27]. As in the training rule of the Boltzmann Machine, $\langle \cdot \rangle_0$ denotes the expectation value of S_{λ_i} when training data are *clamped* to visible neurons and hidden neurons "diffuse" freely until equilibrium is reached. $\langle \cdot \rangle_\infty$ then denotes the expectation value when all neurons diffuse freely until equilibrium is reached.

Let the sigmoid function of neurons have the following general form

$$\varphi_i(x_i) = \theta_L + (\theta_H - \theta_L) \cdot \frac{1}{1 + \exp(-a_i x_i)} \tag{2.22}$$

θ_L and θ_H correspond to the lower and upper asymptotes of the sigmoid function, and a_i is the *activation gain* [27] controlling the slope of the sigmoid function. The energy function for a

symmetric Diffusion Network is defined as [27]

$$E_{DN} = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \frac{\rho_i}{a_i} \int_{s_0}^{s_i} \phi^{-1}(s) ds \tag{2.23}$$

where $\rho_i = 1/R_i$, $s_i = \varphi_i(x_i)$, and $s_0 = \varphi_i(0)$. $\phi(\cdot)$ represents the sigmoid function $\varphi_i(\cdot)$ with $a_i = 1$. Taking the negative derivative of Eq.(2.23) w.r.t. parameter λ_i then gives the system covariates as follows [27].

$$S_{\lambda_i} = s_i \cdot s_j, \qquad \text{for} \quad \lambda_i = w_{ij} \tag{2.24}$$

$$S_{\lambda_i} = \frac{1}{a_i^2} \int_{s_0}^{s_i} \phi^{-1}(s) ds, \qquad \text{for} \quad \lambda_i = a_i \tag{2.25}$$

Substituting Eq.(2.24) into Eq.(2.21) reveals that the training rule for w_{ij} is the same as that of the Boltzmann Machine. Examples of training the symmetric Diffusion Network are given in [27,41].

2.4.4 Discussion

The symmetric Diffusion Network forms a continuous Boltzmann Machine when its drift vector equals the negative gradient of energy. A Diffusion Network in this form is useful in modelling continuous data such as face images [27, 41], and is thus a potential probabilistic model for modelling noisy biomedical data. However, training the Diffusion Network is as computationally-expensive as training the Boltzmann Machine, owing to the necessity of a relaxation search for equilibrium. The diagram of the Diffusion Network's neuron (Fig.2.9), nevertheless, suggests that the diffusion process may be able to be carried out naturally and in parallel in analogue hardware. An example is the Diffusion-Network architecture for hardware implementation proposed by Ting [20], despite that the Diffusion-Network architecture was originally proposed to approximate the binary Boltzmann Machine. The remaining concern

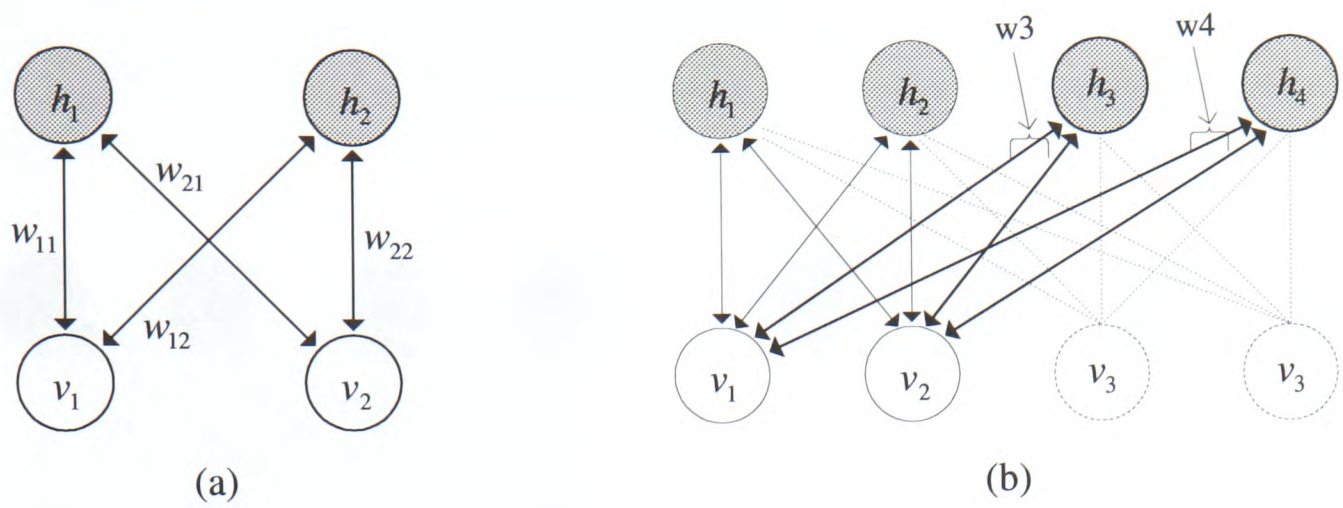


Figure 2.12: *(a)A Restricted Boltzmann Machine with two visible and two hidden neurons (b)A Restricted Boltzmann Machine having comparable representational capability to the Boltzmann Machine in Fig.2.7(b)*

is that massive recurrent connections make analogue implementation costly. The Restricted Boltzmann Machine described in the next section suggests a "restricted" recurrent structure to address this concern.

2.5 The Restricted Boltzmann Machine

2.5.1 Restricted network

The Restricted Boltzmann Machine (RBM) refers to the *Harmonium* model proposed by Smolensky [22]. The RBM employs the same binary stochastic neurons as the Boltzmann Machine, but connects neurons in a "restricted" structure as shown in Fig.2.12(a). Note that w_{ij} denotes the weight connection between visible neuron v_i and hidden neuron h_j. In the RBM, visible and hidden neurons are "restricted" to two separate layers, and connections are "restricted" to inter-layer only. The first restriction enhances the distinctive functions between visible and hidden neurons. As visible neurons represent the features of training data, hidden neurons function as *knowledge atoms* [22] that cooperate to "explain" observed features. This simplifies the

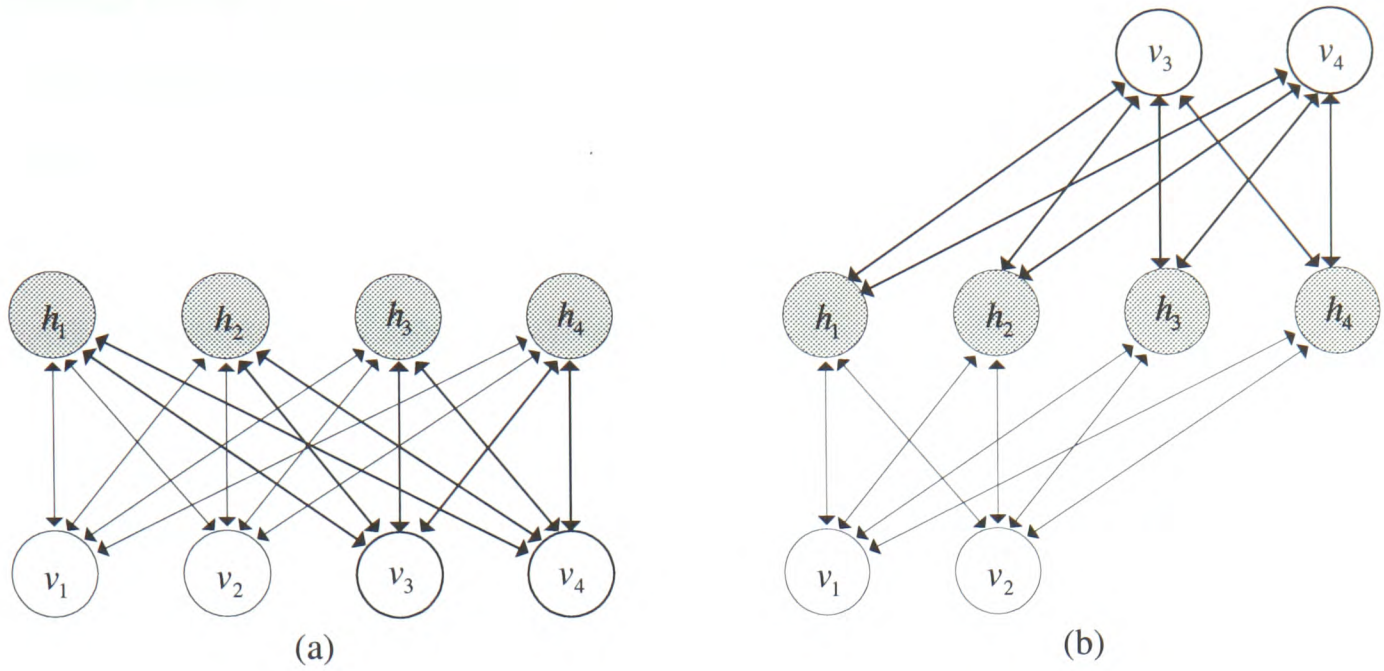


Figure 2.13: *(a)A Restricted Boltzmann Machine with four visible and four hidden neurons (b)A hierarchical network obtained by unfolding visible neurons V3 and V4 to form a third layer on the top*

mathematical analysis of the model, and will be detailed in the next subsection. The second restriction, restricted connections, speeds up the model significantly. The probabilities of the outputs of neurons in the same layer are conditionally independent, given the outputs of neurons in the other layer. Neurons in the same layer can thus be updated in parallel, reducing the time for relaxation search.

Furthermore, the restrictions do not degrade the modelling ability of the RBM significantly. Smolensky shows that the RBM retains a modelling capability comparable to that of the Boltzmann Machine by recruiting extra neurons [22]. The absence of the connection between visible neurons in Fig.2.12(a), for example, can be compensated by the connections $\mathbf{w}^{(3)}$ and $\mathbf{w}^{(4)}$ to the two extra hidden neurons in Fig.2.12(b). Extra visible neurons may be also incorporated to model complicated data, representing more detailed fragments of data features. Furthermore, the two-layer structure is capable of learning a hierarchical representation without pre-defining a particular structure with limited flexibility. The RBM in Fig.2.13(a), for example, can be

unfolded into the equivalent hierarchical network in Fig.2.13(b). The RBM therefore points towards a simpler hardware implementation of recurrent networks without degrading modelling ability.

2.5.2 Multiplicative mixture of experts

The equilibrium form and the training rule of the Boltzmann Machine are inherently applicable to the RBM, because "restricting connections" is equivalent to setting particular connections (e.g. w_{ii}) to zero in the Boltzmann Machine. It is of interest that the equilibrium form of the RBM can be further simplified, indicating that the RBM is a *multiplicative mixture of experts* [42]. Let the vector $\mathbf{v} = \{v_i\}$ denote the state of all visible neurons' outputs (called the *visible state* in this thesis). By viewing each hidden neuron as one "expert", Freund and Haussler show that the equilibrium form of a RBM, given its weight connections $\mathbf{W} = \{w_{ij}\}$, is [42]

$$Pr(\mathbf{v} \mid \mathbf{W}) = \frac{1}{Z}\Pi_i \left(1 + e^{\mathbf{w}^{(i)}.\mathbf{v}}\right) \tag{2.26}$$

where Z is a normalising factor to ensure $\sum_{\mathbf{v}} Pr(\mathbf{v} \mid \mathbf{W}) = 1$, and $\mathbf{w}^{(i)}$ represents the connections between each hidden neuron i and all visible neurons. Eq.(2.26) indicates that each hidden neuron (expert) i is characterised by its *knowledge vector* (in Smolensky's words), $\mathbf{w}^{(i)}$, and that all experts co-operate in a multiplicative form to "influence" the equilibrium distribution of the model [6] [42]. Expert h_i assigns a high probability to a visible state $\mathbf{v}_\alpha$ only when $\mathbf{v}_\alpha$ tends to align itself with the direction of $\mathbf{w}^{(i)}$. The influence strength of each expert h_i depends on the length of its knowledge vector $\|\mathbf{w}^{(i)}\|$. The longer the length, the stronger the influence.

Consider the RBM in Fig.2.14(a) as an example. The possible states of the model correspond to the four corners of the plane space in Fig.2.14(b). Let the *knowledge vector* $\mathbf{w}^{(3)}$ of

[6] The word "influence" is to imply the fact that Freund and Haussler generalised the Restricted Boltzmann Machine into the *Influence Combination Model.*

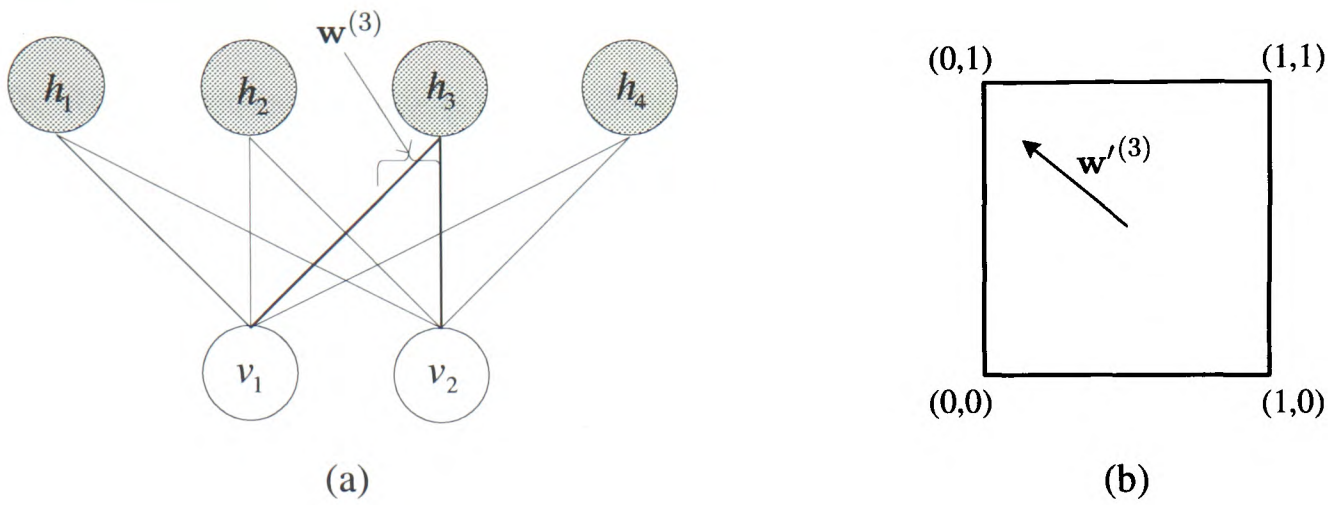


Figure 2.14: *(a)The Restricted Boltzmann Machine with two visible and four hidden neurons. The hidden neuron h3 is characterised by its weight vector* $\mathbf{w}^{(3)}$. *(b)*$\mathbf{w}^{(3)}$ *normalised into the space of visible states*

expert h_3 be normalised as shown in Fig.2.14(b) such that the normalised vector $\mathbf{w}'^{(3)}$ preserves the direction of $\mathbf{w}^{(3)}$ [7]. If the length of $\mathbf{w}^{(3)}$ is much longer than that of all other $\mathbf{w}^{(i)}$, the visible state $(0,1)$ has higher probability than the other three states when the model is in equilibrium. This result is intuitively obvious, as the binary vector $(0,1)$ aligns with $\mathbf{w}^{(3)}$ better than the binary vectors of the other states do. From another point of view, with visible neurons in state $(0,1)$, neuron h_3 has higher "firing" probability (equal to 1) than other neurons, as indicated by Eq.(2.7). This feature is especially useful for tasks such as pattern recognition. Assume the model in Fig.2.14 is trained to recognise the pattern $(0,1)$. When test patterns are presented one-by-one to visible neurons, the responses (knowledge) of expert h_3 indicate directly the presence of $(0,1)$. Ch.4 will show that this feature can be utilised to detect abnormal heartbeats.

[7] The origin of $\mathbf{w}^{(i)}$ is set at (0.5,0.5) because the expectation value of a neuron's output equals to 0.5 when $\mathbf{w}^{(i)} = 0$, as illustrated by Fig.2.7(a)

2.5.3 Discussion

The RBM introduces simplicity to both software and hardware implementation, while retaining comparable modelling ability to the Boltzmann Machine. In addition, the RBM possesses a *multiplicative* form of equilibrium probability distribution. Each hidden neuron functions as an "expert" whose weight vector encodes a particular feature of the modelled data distribution, and hidden neurons (experts) cooperate to "explain" the data observed by the model. This feature not only provides a better understanding of the model's equilibrium, but also suggests an alternative way of carrying out tasks such as pattern recognition with the model. It is notable that the multiplicative form of distribution does not hold for the Boltzmann Machine, owing to the existence of the connections between hidden neurons [8]. Hinton proposed the Product of Experts [23] and further explored the advantages of multiplicative mixture models. One attractive advantage is the *Minimising-Contrastive-Divergence* (MCD) method [24], an approximate but more computationally-efficient training method that avoids the relaxation search for equilibrium. The MCD method is described and discussed in the next section.

2.6 Minimising-Contrastive-Divergence Training

2.6.1 Minimising Contrastive Divergence

The intuitive and correct way to train the equilibrium probability distribution of a generative model is by minimising the KL-divergence between the equilibrium distribution and the distribution of training data. This method was used to derive the training rules for the Boltzmann Machine and the Diffusion Network in previous sections. These training rules inevitably require certain statistics of the equilibrium distribution, in order to calculate the KL-divergence and to

[8] The probability of visible states in Eq.(2.26) is obtained by marginalising the probability in Eq.(2.10) over all possible hidden states. If the connections between hidden neurons exist, the marginal probability won't be a multiplicative form

update parameters correctly, as indicated by Eq.(2.12) and Eq.(2.21). However, collecting the equilibrium statistics is computationally-expensive. If a generative model has fully-connected neurons, only one neuron can be updated at a time in software simulation. As *one full step of Gibbs sampling* refers to updating all neurons once, reaching equilibrium demands many steps of Gibbs sampling.

The *Minimising-Contrastive-Divergence* (MCD) method [24] avoids such expensive computation by minimising the *contrast* (difference) between the following two KL-divergences. (1) The KL-divergence between the equilibrium distribution and the distribution of training data. (2) The KL-divergence between the equilibrium distribution and the distribution of "one-step" Gibbs sampled data [9] [24]. Let P^{∞} denote the equilibrium distribution of a generative model, P^0 the distribution of training data, and P^1 the distribution of one-step Gibbs sampled data. Let $P^A \parallel P^B$ represent the KL-divergence (as defined in Eq.(2.11)) between distributions A and B. The contrast (difference) between the two KL-divergences above is denoted as

$$D = P^0 \parallel P^{\infty} - P^1 \parallel P^{\infty} \tag{2.27}$$

To minimise the contrastive divergence D in Eq.(2.27), the training rule for any parameter λ_i of the generative model follows the negative derivative of D with respect to λ_i as [24]

$$\triangle\lambda_i \propto -\frac{\partial D}{\partial \lambda_i} = -\frac{\partial}{\partial \lambda_i}\left(P^0 \parallel P^{\infty} - P^1 \parallel P^{\infty}\right) \tag{2.28}$$

Fig.2.15 depicts the concept of minimising contrastive divergence in terms of one-dimensional distance. As P^1 is one step closer to P^{∞} than P^0, P^1 is placed between P^0 and P^{∞}. D_X and D_Y correspond to the two KL-Divergences, $P^0 \parallel P^{\infty}$ and $P^1 \parallel P^{\infty}$, respectively. Minimising contrastive divergence thus refers to minimising $D_Z = (D_X - D_Y)$. D_Z is never negative and

[9] One-step Gibbs sampled data refer to the sampled outputs of all neurons obtained by one full step of Gibbs sampling

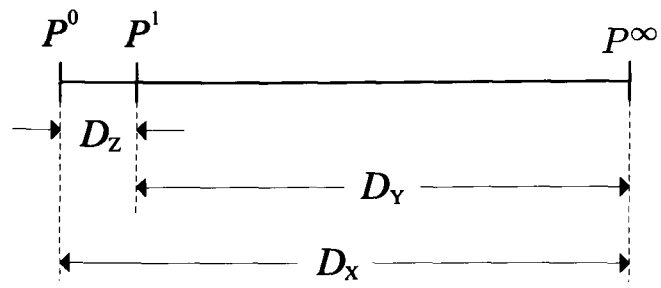


Figure 2.15: *The concepts of "Minimising Contrastive Divergence" in terms of one-dimensional distance*

is only zero when $P^0 = P^1$. However, $P^0 = P^1$ implies $P^0 = P^\infty$, as Gibbs sampling from a non-zero probability should always lead the model to "move" towards equilibrium. The contrastive divergence is thus zero only when the equilibrium distribution of the model is equal to the distribution of training data.

Applying the MCD method to the training rule of the Boltzmann Machine (Eq.(2.12)) removes the need for a full relaxation search for equilibrium. As the training rule in Eq.(2.12) is derived by minimising the KL-divergence $P^0 \parallel P^\infty$, the negative derivative of $P^0 \parallel P^\infty$ w.r.t. w_{ij} is proportional to

$$-\frac{\partial \left(P^0 \parallel P^\infty\right)}{\partial w_{ij}} \propto \langle s_i s_j \rangle_0 - \langle s_i s_j \rangle_\infty \tag{2.29}$$

If the one-step Gibbs sampled data is viewed as another set of training data, minimising the KL-divergence $P^1 \parallel P^\infty$ w.r.t. w_{ij} is

$$-\frac{\partial \left(P^1 \parallel P^\infty\right)}{\partial w_{ij}} \propto \langle s_i s_j \rangle_1 - \langle s_i s_j \rangle_\infty \tag{2.30}$$

where $\langle \cdot \rangle_1$ denotes the expectation value over one-step Gibbs sampled data. Substituting Eq.(2.29) and Eq.(2.30) into Eq.(2.28) gives the MCD training rule for w_{ij} as

$$\begin{aligned} \Delta w_{ij} &\propto \left(\langle s_i s_j \rangle_0 - \langle s_i s_j \rangle_\infty\right) - \left(\langle s_i s_j \rangle_1 - \langle s_i s_j \rangle_\infty\right) \\ &= \langle s_i s_j \rangle_0 - \langle s_i s_j \rangle_1 \end{aligned} \tag{2.31}$$

The terms of equilibrium statistics cancel each other as a result of the MCD method. The MCD training rule thus merely requires one-step Gibbs sampled data, greatly speeding up the training process of generative models.

2.6.2 The Product of Experts

Hinton proposed the Product of Experts, a general form of multiplicative mixture model [23], and showed that the MCD method is effective for training the Product of Experts [24]. Let λ_m represent all the parameters of the m-th simple model, and $p_m(\mathbf{v} \mid \lambda_m)$ denote the probability of data $\mathbf{v}$ under the model m. The Product of Experts combines the "opinions" of N simple models (experts) in a *product* form of

$$P(\mathbf{v} \mid \lambda_1, \ldots, \lambda_N) = \frac{\prod_m p_m(\mathbf{v} \mid \lambda_m)}{\sum_{\mathbf{c}} \prod_m p_m(\mathbf{c} \mid \lambda_m)} \tag{2.32}$$

where $\mathbf{c}$ indexes all possible vectors in the data space, and $P(\mathbf{v} \mid \lambda_1, \ldots, \lambda_N)$ denotes the probability of data vector $\mathbf{v}$ under the multiplicative mixture model. The sum over all possible data in the denominator is to ensure $\sum_{\mathbf{c}} P(\mathbf{c} \mid \lambda_1, \ldots, \lambda_n) = 1$.

Comparing Eq.(2.32) to Eq.(2.26) reveals that the Restricted Boltzmann Machine (RBM) is a particular Product of Experts. While the RBM is a binary stochastic model, Hinton modifies the RBM into *the Product of Experts in the RBM form* (abbreviated as "the PoE/RBM" in the rest of this thesis), and shows that the PoE/RBM is able to model continuous data, specifically, grey-level hand-written digits in [24] [10]. Murray further suggested that the PoE/RBM together with MCD training is amenable to hardware implementation [25]. The PoE/RBM is therefore of great interest.

The PoE/RBM has the same network structure as the RBM (Fig.2.12). In order to model

[10] Each pixel of the images of grey level hand-written digits is an integer ranging from 0 to 255, and thus approximates continuous-valued data

continuous-valued data, Hinton introduces two approximations to the RBM. The first approximates the post-sigmoid probabilities of visible neurons directly as the outputs of visible neurons. If v_i represents the output of visible neuron i,

$$\begin{aligned} v_i &= Pr(v_i = 1 \mid \{h_j\}) \\ &= \sigma(\sum_j w_{ij} h_j) = \frac{1}{1 + \exp(-\sum_j w_{ij} h_j)} \end{aligned} \tag{2.33}$$

where h_j represents the output of hidden neuron j. This approximation is equivalent to "removing the sampling action" from the neuron in Fig.2.7(a). The visible neurons of the PoE/RBM thus become *continuous deterministic*, while the hidden neurons of the model remain *binary stochastic* with a probability

$$Pr(h_i = 1 \mid \{v_j\}) = \frac{1}{1 + \exp(-\sum_j w_{ij} v_i)} \tag{2.34}$$

To train such a combination of continuous deterministic and binary stochastic neurons, the second approximation is in the use of the MCD training (Eq.(2.31)). The outputs of neurons in Eq.(2.31), s_i and s_j , are replaced by the post-sigmoid probabilities of neurons. Let pv_i and ph_j denote $Pr(v_i = 1 \mid \{h_j\})$ and $Pr(h_j = 1 \mid \{v_i\})$, respectively. The MCD training rule for the PoE/RBM is written as the following

$$\Delta w_{ij} \propto \langle pv_i \cdot ph_j \rangle_0 - \langle pv_i \cdot ph_j \rangle_1 \tag{2.35}$$

Let $\{\mathbf{v}^0\}$ represent a set of training data. The training process for the PoE/RBM is summarised as follows, and illustrated in Fig.2.16.

1. *Clamp* each training datum $\mathbf{v}^0$ to the visible neurons, i.e. setting $\mathbf{v}^0$ as the initial visible state .

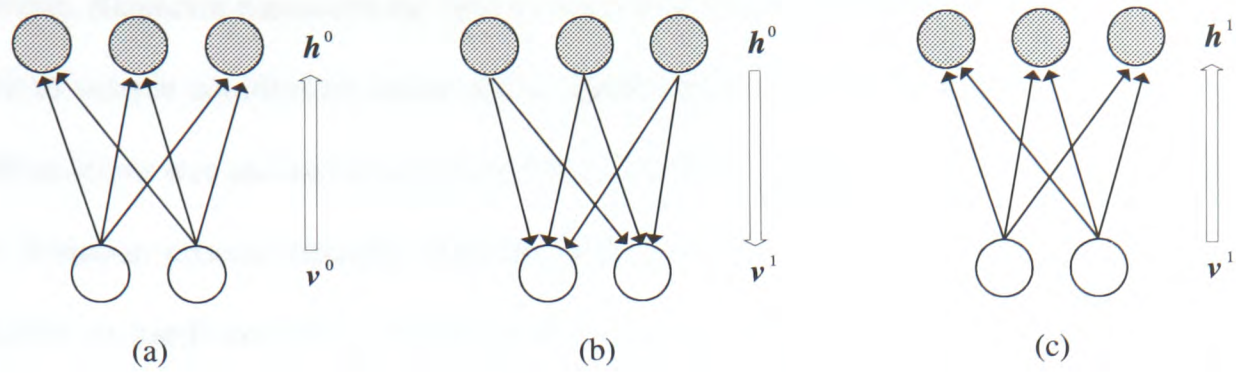


Figure 2.16: *One-step Gibbs sampling the PoE/RBM with two visible and three hidden neurons (a)Given initial visible state* $\mathbf{v}^0$*, sample* $\mathbf{h}^0$ *(b)Given* $\mathbf{h}^0$*, sample the one-step reconstructed visible state* $\mathbf{v}^1$ *(c)Given* $\mathbf{v}^1$*, sample the corresponding one-step hidden state* $\mathbf{h}^1$

2. Given $\mathbf{v}^0$, compute the probabilities $\{ph_j^0\}$ of all hidden neurons according to Eq.(2.34), and then sample a binary value h_j^0 for each hidden neuron (Fig.2.16(a)).

3. Given the sampled hidden state $\mathbf{h}^0 = \{h_j^0\}$, compute the probabilities $\{pv_i^1\}$ according to Eq.(2.33). Set $v_i^1 = pv_i^1$ and let $\mathbf{v}^1 = \{v_i^1\}$ represent the *one-step reconstructed* visible state (Fig.2.16(b)).

4. Given $\mathbf{v}^1$, obtain $\{ph_j^1\}$ and the one-step Gibbs sampled hidden state $\mathbf{h}^1$ (Fig.2.16(c)).

5. Update all connection weights $\{w_{ij}\}$ according to Eq.(2.35). Note that $\mathbf{pv}_i^0 = \mathbf{v}^0$.

6. Repeat step (1)−(5) till weight changes are negligible.

2.6.3 Discussion

The MCD method introduces an efficient approach to training generative models, whereby a relaxation search for equilibrium is avoided [11]. Although relaxation search is not costly in analogue hardware, it is still essential to know the time to reach equilibrium [20, 38]. For

[11] Note that the MCD training rule for the Mean-Field Boltzmann Machine is also proposed in [43] to further enhance software simulation of the Boltzmann Machine. The limitation of Mean-Field Approximation described in Sec.2.3.3 however excludes it from the interest of this research.

example, Alspector measured the time to reach Boltzmann equilibrium in [38], in order to know when to sample equilibrium states and to update parameters. The settling time for solving the XOR problem was shown to vary from 130 to 1,700 ns in [38]. Ting also showed that the speed of a diffusion process strongly depends on the parameters of neurons and the voltage range available in hardware [20]. The time required for reaching equilibrium thus varies between networks and with hardware implementation. One-step Gibbs sampling in hardware, on the contrary, always takes four steps (clock cycles). The "clock frequency" in hardware depends upon the trade-off between speed and power consumption. Even if one-step sampling is not faster than a natural diffusion process in analogue hardware, the MCD training rule at least provides a simple control over timing without the need to know the time to reach equilibrium. The MCD training method is therefore a favourable choice.

The PoE/RBM, together with its MCD training rule, suggests a probabilistic model that is not only computationally-efficient, but is also able to model continuous-valued data, for example hand-written digits in Hinton's experiments [24, 44]. The restricted structure, furthermore, enhances the simplicity of the PoE/RBM [25]. However, it is notable that the PoE/RBM actually consists of *binary stochastic* and *continuous deterministic* neurons, rather than *continuous-valued stochastic* neurons. The lack of real continuous-valued probabilistic behaviour is likely to limit the representational capability of the PoE/RBM, especially when continuous-valued data of interest are complex. This question will be examined and answered in Ch.3.

2.7 Summary for probabilistic neural computation

Figure 2.17 shows the relationship between the recurrent neural networks reviewed in this thesis as a tree diagram. The behavioural characteristics of each model decide its position in the tree diagram. The characteristics are distinguished between deterministic and probabilistic,

between binary- and continuous-valued, and between full and restricted connection. In this family, probabilistic models possess modelling ability and flexibility that is superior to that of deterministic models. Continuous-valued models are undoubtedly more suitable for modelling continuous data than are binary-valued models. Restricted connection, furthermore, renders a probabilistic model's architecture simple and retains comparable modelling ability to models with full inter-connections.

Though not discussed in previous sections, the Cellular Neural Network (CNN) developed by Chua [45–47] is a variant of the continuous Hopfield Network, in which each neuron is "restricted" to have connections to its neighbouring neurons only. As such restriction invariably enhances the hardware amenability of the Cellular Neural Network, its well-developed hardware in [30, 48] is an example of exploiting a natural diffusion process (without noise) in analogue circuits. The Cellular Neural Network is therefore also included, although this research mainly concerns probabilistic models. The PoE/RBM is a network of both binary stochastic and continuous deterministic neurons. The PoE/RBM is put next to the RBM as a variant of it, rather than under the branch of continuous probabilistic models. This indicates the need for more simulations to examine the capability of the PoE/RBM in modelling continuous data. The dashed link, furthermore, stresses the existence of continuous deterministic neurons. Finally, the question mark highlights the area this thesis explores, a continuous probabilistic model which is not only able to model continuous data but also simple and amenable to hardware. Ch.3 will discuss the capability of the PoE/RBM, and Ch.4 will then provide a possible solution to fill the box.

Several recurrent neural networks in Fig.2.17 have been successfully implemented in VLSI, including the Hopfield Network [13], the Cellular Neural Network [30, 48, 49], and the Boltzmann Machine [19, 38, 39, 50]. Minimising-Contrastive-Divergence (MCD) training, as discussed in Sec.2.6, is also demonstrated in VLSI in [51–53]. As this research intends to identify

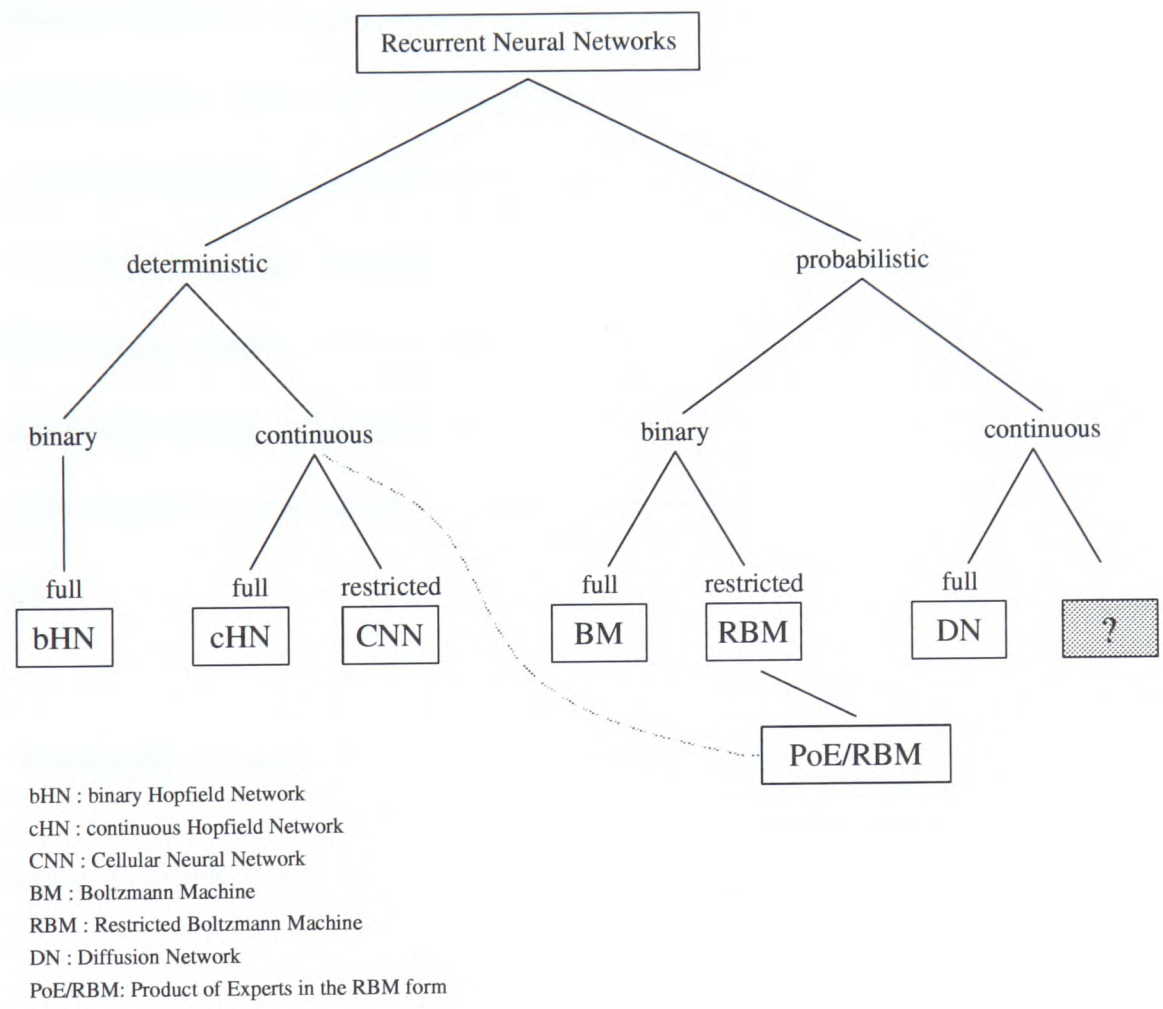


Figure 2.17: *The characteristics of the recurrent neural networks reviewed in the thesis*

a continuous probabilistic model from the family of recurrent neural networks, the following sections review the VLSI implementation of the Boltzmann Machine and MCD training rule.

2.8 The Boltzmann Machine in VLSI

The Boltzmann Machine has been implemented in VLSI, while the Diffusion Network and its continuous stochastic neuron have not. Nevertheless, the Boltzmann Machine in VLSI in [19, 38, 39, 50] uses artificially-generated noise to realise binary stochastic neurons in VLSI, resulting in VLSI neurons with intrinsically continuous-stochastic behaviour. The Boltzmann Machine in VLSI thus also suggests a way to implement continuous-stochastic neurons in VLSI. This section mainly reviews Alspector's Boltzmann-Machine (BM) neurons in VLSI, and discusses the design techniques and considerations relating to the VLSI neurons. The VLSI implementation of the training rule for the Boltzmann Machine in Eq.(2.12) will also be introduced.

2.8.1 Stochastic neuron in VLSI: Block diagram

The neuron of the Boltzmann Machine is defined in Eq.(2.7), repeated here as Eq.(2.36).

$$Pr(s_i = 1 \mid x_i) = \frac{1}{1 + \exp(-x_i/T)} \tag{2.36}$$

where $x_i = \sum_j w_{ij} s_j$ is the total input to the neuron, and T is the "temperature factor". The VLSI neuron implemented by Alspector performs the following computation [19]

$$s_i = \varphi\left(\beta \cdot (x_i + n_i)\right) \tag{2.37}$$

where $\varphi(\cdot)$ represents a sigmoid function such as $tanh(\cdot)$, and β is the gain factor, controlling the slope of the sigmoid function [12]. The input n_i is a Gaussian-distributed noise with zero mean , while $x_i = \sum_j w_{ij}s_j$ still represents the deterministic inputs to the neuron. Eq.(2.37) indicates that s_i is continuous-valued, and the Gaussian-noise input causes s_i to be probabilistic. In hardware, s_i is passed through a comparator, resulting in a binary-valued output $s_i^B = sgn(s_i)$. It is shown in [19] that the probability density of s_i^B approximates the logistic function in Eq.(2.36). s_i^B therefore represents the output of a binary stochastic neuron in the Boltzmann Machine in VLSI.

The main advantage of this VLSI neuron is that both simulated annealing and relaxation for equilibrium become efficient in hardware. The VLSI neurons actually produce continuous-valued outputs, s_i, in hardware [39]. Relaxation search over binary states is approximated by a continuous-valued diffusion process in continuous time. The use of analogue circuits further facilitates real-time computation, enabling all VLSI neurons to be updated simultaneously. The time for relaxation search is thus reduced in hardware. In addition, Eq.(2.37) indicates that the width (variance) of noise is proportional to the temperature T in Eq.(2.36). The increase of noise variance weakens the influence of x_i on s_i and thus on s_i^B effectively. Therefore, a simulated annealing process can be carried out by reducing noise variance gradually in hardware. It is shown in [38] that the Boltzmann Machine in VLSI solves an XOR problem 100,000 times faster than the Boltzmann Machine simulated on a digital computer.

Fig.2.18 shows the block diagram of a BM neuron in VLSI. The multipliers calculate $(w_{ij} \cdot s_j)$, outputting a total current I_{in} proportional to $\sum_j w_{ij}s_j$. The I-V converter then converts I_{in} into a voltage, V_{in}, for the input of a summing amplifier. The reference zero of a voltage is defined by V_{ref}. The summing amplifier not only adds controlled noise to the input V_{in}, but also

[12]The gain factor β is included for simulating a Mean-Field Boltzmann Machine [54], in which noise inputs are removed and β is adapted in proportion to $1/T$. Therefore, the gain factor β is kept constant when simulating a Boltzmann Machine.

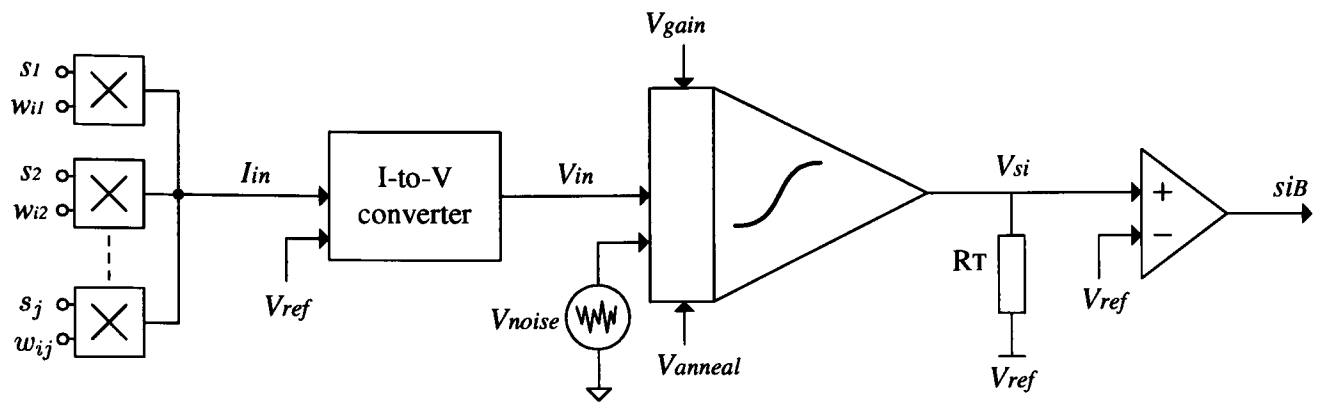


Figure 2.18: *The block diagram of a BM neuron in VLSI.*

introduces a sigmoidal nonlinearity. The noise input V_{noise} is controlled by V_{anneal}, while V_{gain} controls the slope of the sigmoidal nonlinearity. As a result, the summing amplifier outputs a current proportional to $\varphi(\beta \cdot (x_i + n_i))$, and the resistor R_T transforms the current into a voltage, V_{si}, which represents s_i in Eq.(2.37) and is sent to the inputs of all other neurons. Finally, the comparator samples V_{si} into a binary output, representing the states of the Boltzmann Machine. The design techniques and considerations for individual circuit blocks are discussed in the following subsection.

2.8.2 Stochastic neuron in VLSI: design techniques and considerations

This section discusses the design techniques and considerations of the multiplier, the current-to-voltage converter, the sigmoid-function circuit, and the noise generator in Fig.2.18.

Multiplier

While a large variety of multipliers has been proposed, multipliers with a simple architecture and a current-mode output are preferable for the implementation of a VLSI neuron. A simple architecture is essential for reducing the power- and area- consumption of a VLSI neuron, especially when a neuron has many inputs. A current-mode output, furthermore, eases the calculation of $\sum_j w_{ij} s_j$. The value of the sum varies widely, depending on the number of

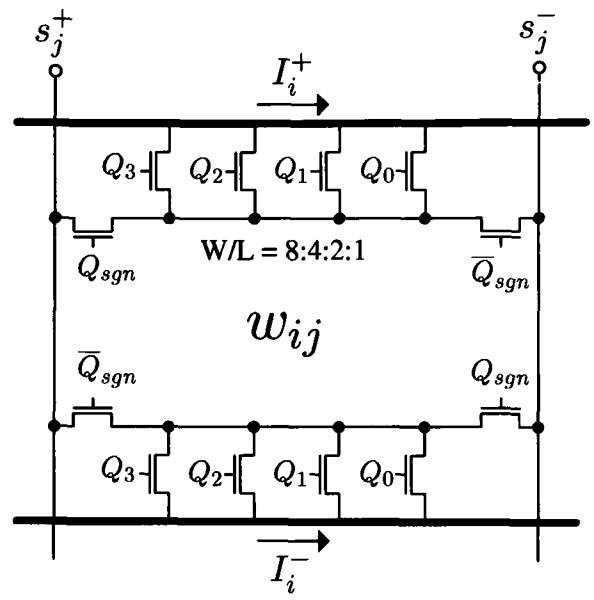


Figure 2.19: *A 5-bit digital-to-analogue converter that is employed as a multiplier for the VLSI neuron in Fig.2.18.*

inputs as well as the values of $\{w_{ij}\}$ and $\{s_j\}$. As a current has a more flexible range than a voltage in VLSI, current-mode representation of $(w_{ij} \cdot s_j)$ offers more flexibility for integrating a large or variable number of inputs.

In a BM neuron in VLSI, each w_{ij} is stored as a 5-bit binary number. Calculating $(w_{ij} \cdot s_j)$ thus requires only a simple digital-to-analogue converter (DAC), as shown in Fig.2.19 [38]. The bit Q_{SGN}, coding the sign of w_{ij}, selects the true or complementary inputs of s_j. The four bits $Q_0 - Q_3$ then control the transistors to sink or source currents, in proportion to the conductance of the transistors. As a result, the differential current, $(I_i^+ - I_i^-)$, is proportional to $(w_{ij} \cdot s_i)$.

Current-to-voltage converter

An I-V converter can be implemented as an amplifier with a feedback resistor, as shown in Fig.2.20. With the negative-feedback resistor, the amplifier provides a low-impedence input for summing up currents from multipliers. The negative input terminal is held at a constant voltage, V_{ref}, which is set by the positive input terminal [55]. The input current I_{in} is thus converted into an output voltage of

$$V_o = V_{ref} - I_{in} \cdot R \tag{2.38}$$

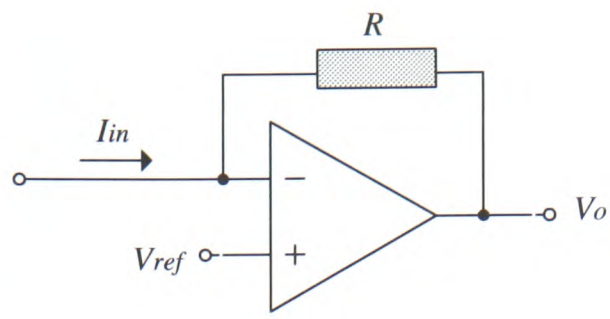


Figure 2.20: *A current-to-voltage converter consisting of an amplifier and a feedback resistor*

Eq.(2.38) indicates that the output voltage V_o is linearly proportional to I_{in}, with V_{ref} defining the reference zero of V_o. In [56], Choi further employs a voltage-controlled resistor as the feedback resistor R [57], in order to normalise the input current I_{in} and thus to confine V_o in a desired range when the amount of input current is large.

Sigmoid-function circuit

If the sigmoidal nonlinearity of a model is not realised as faithfully as possible in hardware, the modelling ability in hardware may be degraded. At least, additional simulations are necessary to estimate and then possibly compensate for any inaccuracy [58].

The most obvious source of a sigmoidal nonlinearity is the exponential current-voltage relationship of transistors [59]. Fig.2.21(a) illustrates the symbol of an N-type metal-oxide-semiconductor (NMOS) transistor. As a MOS transistor is biased in the *subthreshold* region [55] [13], the relationship between its gate-source voltage V_{gs} and drain current I_d is

$$I_d = \frac{W}{L} I_0 e^{\kappa V_{gs}} \tag{2.39}$$

where I_0 and κ are process-dependent constants in a VLSI technology, and W/L specifies the transistor's size. Therefore, if the two transistors in Fig.2.21(b) are biased in the subthreshold

[13] A MOS transistor is biased in the *subthreshold* region when its gate-source voltage is smaller than its threshold voltage (around 0.7-0.9V in a conventional VLSI technology).

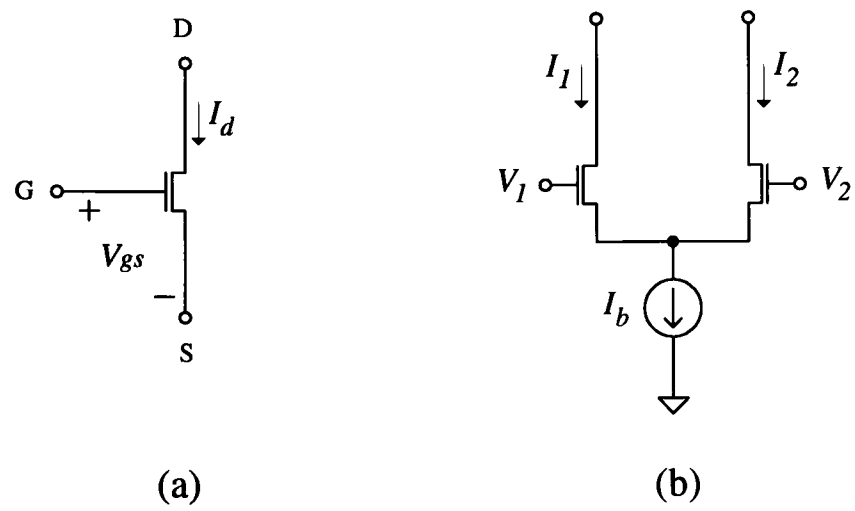


Figure 2.21: *(a) The symbol for a NMOS transistor. (b)The circuit diagram of a differential pair comprised of two NMOS transistor. I_b is the bias current of the differential pair.*

region, a differential input voltage $(V_1 - V_2)$ will induce a differential current $(I_1 - I_2)$ of [59]

$$I_1 - I_2 = I_b \tanh \frac{\kappa(V_1 - V_2)}{2} \tag{2.40}$$

where I_b is the bias current of the differential pair. Eq.2.40 reveals that the relationship between $(V_1 - V_2)$ and $(I_1 - I_2)$ is exactly the sigmoidal nonlinearity of an artificial neuron.

Fig.2.22 shows the circuit diagram of the summing amplifier designed for a BM neuron in VLSI (Fig.2.18) [39]. Transistors M1-M6 are all biased in the subthreshold region. As the input voltage $(V_{in} - V_{ref})$ represents the total deterministic input to a neuron, i.e. x_i in Eq.(2.37), the differential pair, M1-M2, implements the sigmoid nonlinearity, transforming $(V_{in} - V_{ref})$ into a differential current $(I_1 - I_2)$ according to Eq.(2.40). The two differential pairs composed of M3-M6 then multiply $(I_1 - I_2)$ by the differential voltage V_{gain}, resulting in a new differential current $(I_3 - I_4)$ according to [59]

$$I_3 - I_4 = I_b \tanh \frac{\kappa(V_{in} - V_{ref})}{2} \tanh \frac{\kappa V_{gain}}{2} \tag{2.41}$$

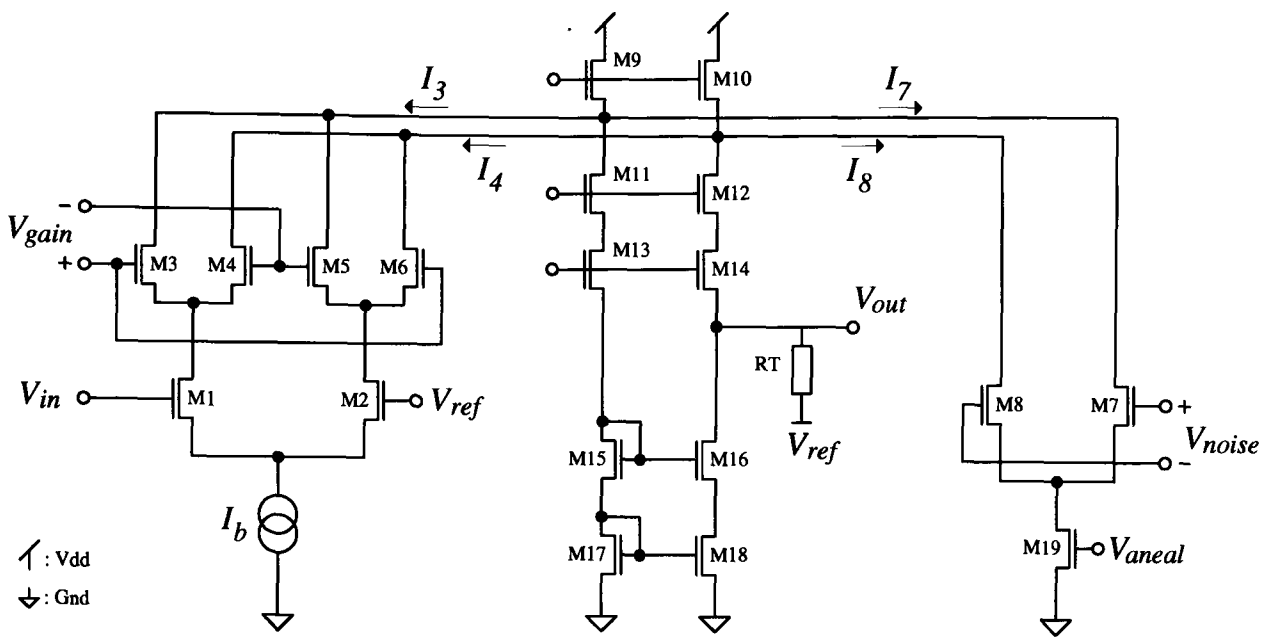


Figure 2.22: *The circuit diagram of the summing amplifier used for the VLSI neuron. The symbols for ground and Vdd will be used in all circuit diagrams in this thesis.*

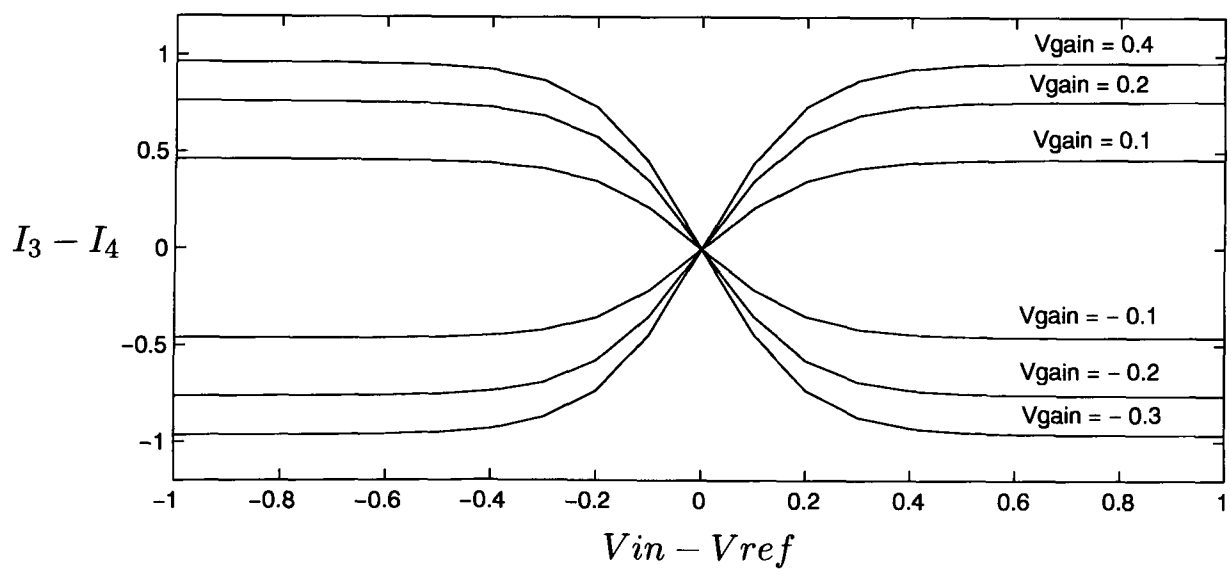


Figure 2.23: *The current-voltage relationship according to Eq.(2.41) ($I_b = 1$, $\kappa = 10$).*

Fig.2.23 illustrates the current-voltage relationship in Eq.(2.41). Obviously, the relationship between $(I_3 - I_4)$ and $(V_{in} - V_{ref})$ is sigmoidal, and V_{gain} adapts its slope. V_{gain} therefore corresponds to the gain factor β in Eq.(2.37). Regarding the noise input V_{noise}, the differential pair M7-M8 converts V_{noise} into a differential current $(I_7 - I_8)$, and V_{anneal} adjusts the width (variance) of noise currents via M19. The deterministic current $(I_3 - I_4)$ and the noise current $(I_7 - I_8)$ are then summed up by the summing circuit M9-M18, producing an output current corresponding to s_i in Eq.(2.37).

Noise generator

The primary consideration is that the noise inputs for any two neurons should be uncorrelated. If noise inputs are correlated, the correlation will be "collected" by the training rule in Eq.(2.12), and will subsequently introduce training errors. Alspector proposed an efficient implementation of multiple uncorrelated noise sources, basing on a linear-feedback shift register (LFSR) [21]. Fig.2.24 shows the block diagram of the proposed noise generator. The D-type flip-flops together with the exclusive-OR (XOR) feedback gate constitute an N-stage LFSR. The LFSR generates a pseudo-random bit stream with a maximal length of $(2^N - 1)$ [21]. The outputs of the flip-flops are tapped to the 3-input XOR gates, according to a tap-pattern table in [21], for obtaining K channels of pseudorandom bit streams. These pseudorandom bit streams are uncorrelated for a length of $(2^N - 1)/K$ bits. Passing the pseudorandom bit streams through low-pass filters then gives K channels of analogue noise with a Gaussian-like distribution [21].

It is important to ensure that analogue noise inputs to neurons are uncorrelated during each training step. Let f_c be the clock frequency of CK_n for the noise generator in Fig.2.24. The K

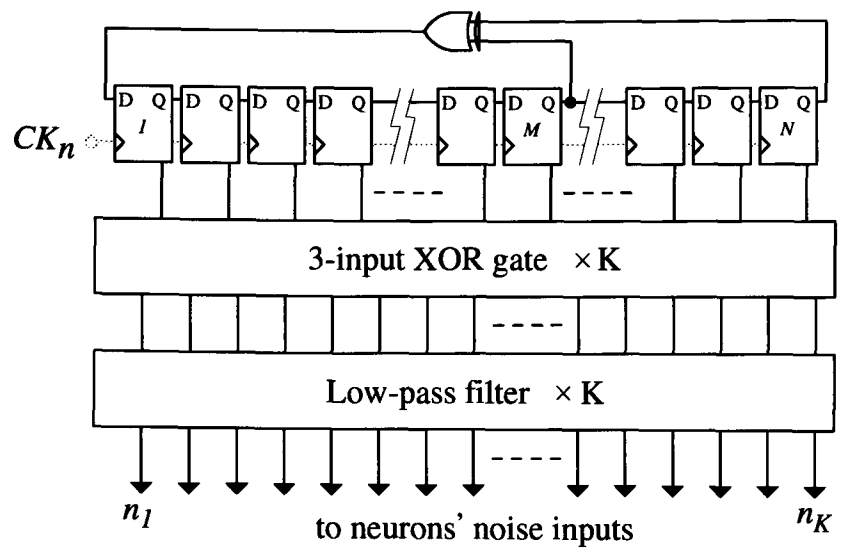


Figure 2.24: *The block diagram of the noise generator basing on one linear- feedback-shift-register (LFSR).*

channels of analogue noise are uncorrelated for a period of

$$t_u = \frac{\text{(uncorrelated length of pseudorandom bit streams)}}{f_c} \sim \frac{2^N - 1}{K \cdot f_c} \tag{2.42}$$

For training a Boltzmann Machine, t_u should be longer than the time required for annealing and relaxing towards equilibrium at each training step [21]. Following the description of a BM neuron in VLSI, the next subsection describes the VLSI circuits for training a Boltzmann Machine.

2.8.3 Training a Boltzmann Machine in VLSI

In the VLSI implementation of the Boltzmann Machine, parameters are adapted according to the following simplified version [19] of the training rule in Eq.(2.12).

$$\triangle w_{ij} = sgn[(s_i \cdot s_j)^+ - (s_i \cdot s_j)^-] \tag{2.43}$$

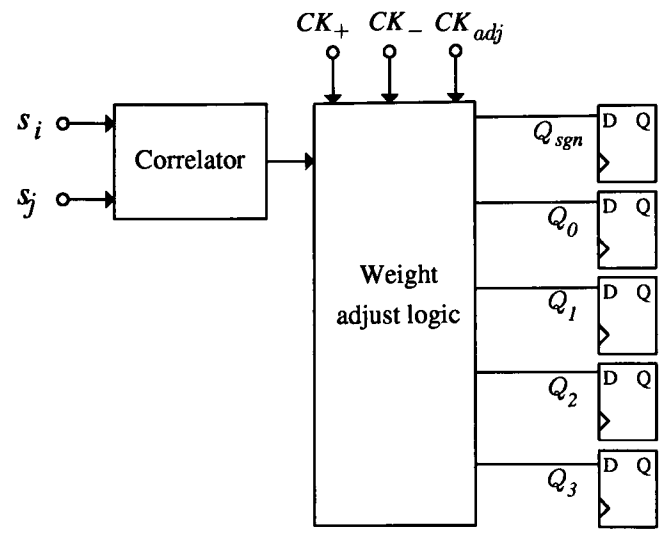


Figure 2.25: *The block diagram of the training circuit for the Boltzmann Machine in VLSI.*

where + and − denote the *clamping* and *free-running* phases, respectively. Eq.(2.43) differs from Eq.(2.12) as follows. Firstly, the correlation $(s_i \cdot s_j)$ is computed for each training datum, as opposed to being averaged over all training data. Secondly, only the sign of the difference between two correlations is taken. The training rule in Eq.(2.43) thus suggests merely an updating direction, not a precise updating value. The simulation results in [50] show that, with this simplified training rule, the Boltzmann Machine is able to learn large problems such as mapping letters to phonemes. Therefore, it is plausible to implement the simplified training rule for training the Boltzmann Machine in VLSI.

Fig.2.25 shows the block diagram of the training circuit [38, 39]. The correlator contains a wide-range Gilbert multiplier [59], calculating $(s_i \cdot s_j)^+$ and $(s_i \cdot s_j)^-$ in Eq.(2.43). After CK_+ and CK_- sample and hold the two correlations, the clock signal CK_{adj} triggers the logic circuit to increase or decrease the weight accordingly. The training circuit in Fig.2.25 is simple and is proved to be reliable in the experiments in [19].

2.8.4 Discussion

The VLSI implementation of the Boltzmann Machine demonstrates that the incorporation of artificially-generated noise is an efficient way of realising binary probabilistic behaviour in VLSI. The BM neuron in VLSI is, however, intrinsically a continuous stochastic neuron, as indicated by Eq.(2.37). The Diffusion Network further uses this form of neuron to reason and to explore continuous-valued probabilistic dynamics [27], as described in Sec.2.4. Compare the block diagram in Fig.2.18 to the neuron of the Diffusion Network in Fig.2.9. It is notable that the VLSI neuron in Fig.2.18 is a particular Diffusion-Network (DN) neuron, i.e. a DN neuron with an infinite resistance and a zero capacitance. While Ting proposed the Diffusion-Network architecture for "approximating" and simulating the Boltzmann Machine in VLSI [20], the greater interests of this research lie in the use of VLSI neurons with noise-induced stochastic behaviour to realise continuous-valued probabilistic computation in VLSI.

However, the VLSI neuron implemented by Alspector is not ideal for exploring continuous-valued probabilistic computation in VLSI. Although 5-bit resolution for $\{w_{ij}\}$ is proved to be enough for the Boltzmann Machine [38], a higher resolution or even continuous-valued $\{w_{ij}\}$ is necessary for modelling the structure of complex, continuous-valued data. Provided that $\{w_{ij}\}$ are continuous-valued in hardware, an analogue multiplier is preferable to the DAC multiplier in Fig.2.19. Fig.2.23 also reveals another non-ideal computation in the VLSI neuron. The output range (asymptotes) of the sigmoid-function circuit is changed with V_{gain}, while ideally V_{gain} should change the slope of the sigmoid function only. This non-ideal effect, though negligible for the binary-valued Boltzmann Machine, may limit the output range of neurons and subsequently the representation power of a continuous-valued model. Therefore, an improved implementation of the sigmoid function is also desirable for exploring continuous probabilistic computation in VLSI.

The VLSI implementation of the Boltzmann Machine also demonstrates that minimising-KL-divergence training becomes efficient in hardware, even though it is time-consuming in software simulation. The efficiency is attributed to the fact that analogue implementation reduces the time for relaxation search significantly. However, the time required for reaching equilibrium in hardware inevitably depends on the tasks, the size of the Boltzmann Machine, and its circuit architecture [20, 38], as discussed in Sec.2.6.3. Minimising-contrastive-divergence (MCD) training, on the contrary, provides simpler control over timing. MCD training therefore compares favourably with minimising KL-divergence training in hardware implementation, in spite of that there may be no significant improvement in computing speed. The following section discusses the MCD training circuit proposed in [52,53].

2.9 Minimising-contrastive-divergence training in VLSI

The minimising-contrastive-divergence (MCD) training rule in Eq.(2.35) has been demonstrated in VLSI in [52, 53]. Although this training rule is used for training the PoE/RBM, the MCD training method is applicable to other probabilistic recurrent networks. The similarity between the training rules of the Boltzmann Machine and the Diffusion Network (Eq.(2.12) and Eq.(2.24), respectively) further implies that the MCD training circuits for other models may employ an architecture similar to that of the PoE/RBM. Therefore, this section describes and discusses the MCD training circuit proposed in [52,53].

2.9.1 Block diagram

As suggested in [25], Fleury implemented a simplified MCD training rule in VLSI rather than the training rule in Eq.(2.35) for the PoE/RBM [52,53]. The simplified training rule is

$$\triangle w_{ij} = \eta_w \cdot sgn\left(\langle s_i \cdot s_j \rangle_0 - \langle s_i \cdot s_j \rangle_1\right) \tag{2.44}$$

where s_i and s_j represent the post-sigmoid probabilities of neurons in the PoE/RBM. Only the sign of the "contrastive divergence" is taken to determine an updating direction. All weights are thus updated directionally by a fixed-size step, η_w. This simplification inherently yields a simpler training circuit for the PoE/RBM. It is notable that a similar approach was adopted to simplify the training circuit for the Boltzmann Machine [19], as described in Sec.2.3.2. The simulation results in [25] show that, with the simplified training rule, the PoE/RBM remains capable of modelling both artificial sensory data and heartbeat data, and the price of the simplicity is a slightly longer training time. This encourages the use of an MCD training circuit according to Eq.(2.44) for the PoE/RBM.

Fig.2.26 shows the block diagram of the MCD training circuit proposed in [52, 53]. s_i and s_j are analogue voltages representing post-sigmoid probabilities of neurons. The analogue multiplier outputs a current proportional to $(s_i \cdot s_j)$. Triggered by CK_+, the current memory samples and holds the current I^+, which represents $(s_i \cdot s_j)$ computed from a single training datum. After one-step Gibbs sampling, the multiplier outputs another current I^-, representing $(s_i \cdot s_j)$ computed from one-step Gibbs-sampled states. The current subtractor subsequently subtracts I^- from I^+, and outputs a current I_{sub}. The current I_{sub} is proportional to the "contrastive divergence" between the training datum and its one-step Gibbs sample. Triggered by the digital control circuit, the accumulator accumulates and averages I_{sub} over several training data. The output current of the accumulator, I_{ave}, thus represents the expected value of the con-

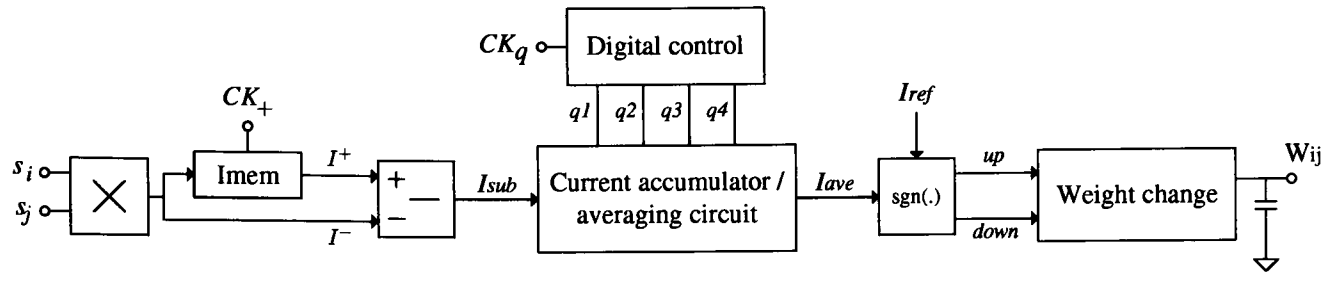


Figure 2.26: *The block diagram of the MCD training circuit proposed in [52, 53].*

trastive divergence, i.e. $\langle s_i \cdot s_j \rangle_0 - \langle s_i \cdot s_j \rangle_1$ in Eq.2.44. Finally, the sign circuit compares I_{ave} to a reference current to obtain an update direction, and the weight-changing circuit updates the weight accordingly.

Unlike the training circuit for the Boltzmann Machine, the MCD training circuit in Fig.2.26 includes an accumulator and stores the weight as an analogue voltage on a capacitor. The accumulator averages contrastive divergences, suggesting a more plausible (i.e. an expected) updating direction. Furthermore, analogue storage of weights enhances the representation of continuous data. It is crucial for the weight-changing circuit to support small voltage changes, such that the training errors induced by the simplification in Eq.(2.44) can be minimised by training weights with a small stepsize, η_w [52]. With the accumulator and weight-changing circuit, the MCD training circuit enhances a model's ability to learn continuous-valued data in VLSI. The following subsection describes and discusses the accumulator and the weight-changing circuit. Details of other component circuits are in [52] and [53].

2.9.2 Design technique and considerations

Current accumulator

The current accumulator proposed in [52] consists of dynamic current mirrors. Fig.2.27 shows one cell of the dynamic current mirror [60]. When the switch is closed, the gate-source voltages of M1 and M2 are forced to be equivalent. The drain current of M2 consequently equals I_{in}.

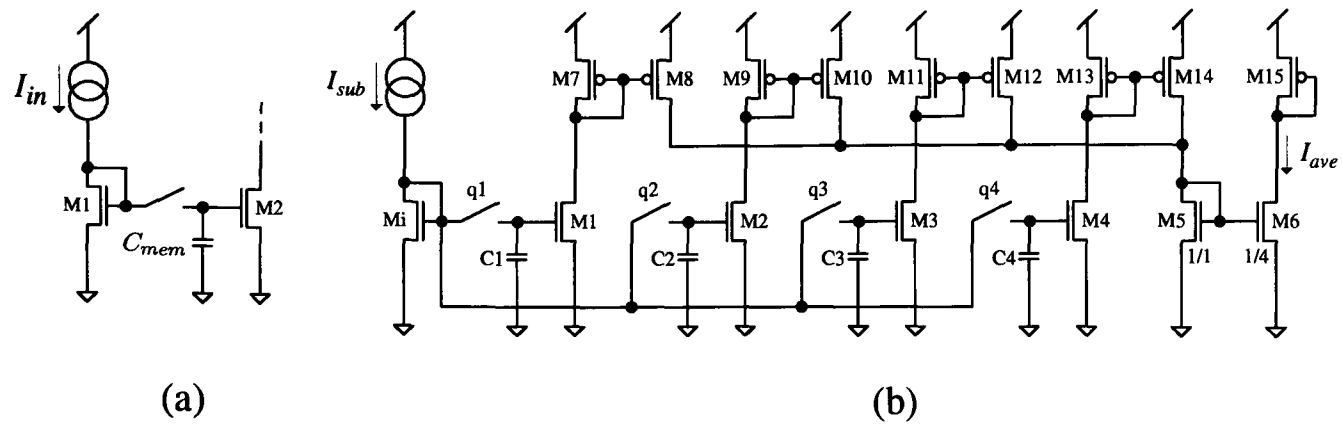


Figure 2.27: *(a)The circuit diagram of a dynamic current mirror (b)The circuit diagram of a current accumulator for the training circuit in Fig.2.26.*

At the same time, the capacitor C_{mem} stores the gate-source voltage of M2. Therefore, the dynamic current mirror samples and holds a value of I_{in} every time the switch is closed. Note that the current memory in Fig.2.26 is implemented as one dynamic current mirror [53].

Fig.2.27 shows the circuit diagram of the current accumulator proposed in [52], consisting of four dynamic current mirrors. I_{sub} represents the output current from the current subtractor in Fig.2.26. The digital signals q1-q4 close the switches, one at a time. After the four switches are closed sequentially and individually, four values of I_{sub} are sampled and held. The four current mirrors, M7-M14, then duplicate the four I_{sub} and sum the four duplicated currents into the drain of M5. As the transistor-size ratio between M5 and M6 is $4:1$, the drain current of M6, I_{ave}, equals the average of the four I_{sub}.

The accumulator in Fig.2.27 reveals a tradeoff. Although the expected value of the contrastive divergence in Eq.(2.44) is better estimated by averaging over a large number of training data, accumulating a large number of training data is area-consuming in hardware. Therefore, a compromise must be made between accuracy and hardware area. The accumulator in Fig.2.27 averages over only four training data, and thus introduces estimation errors. It is important to take the estimation errors into account in simulation, to make sure that the errors do not prevent a model in VLSI from modelling data successfully.

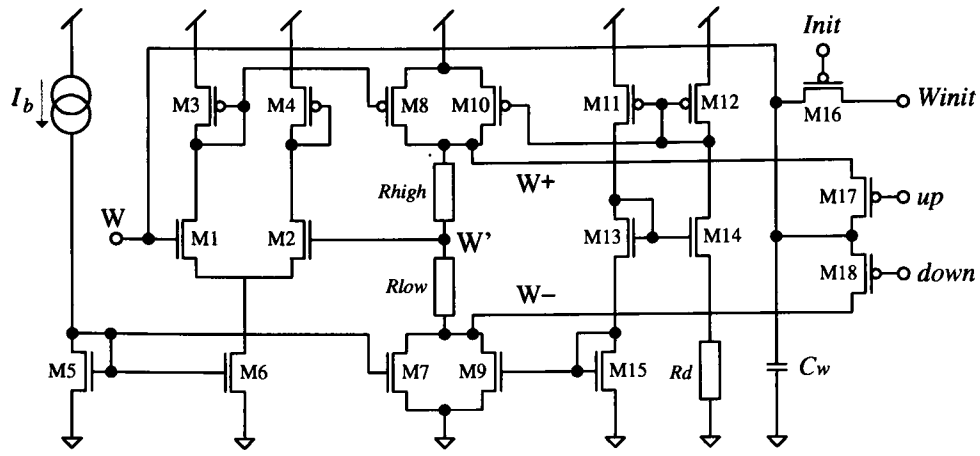


Figure 2.28: *The weight-changing circuit for the MCD training circuit in Fig.2.26.*

Weight-changing circuit

The main challenge for a weight-changing circuit is to modify a weight voltage with as small a step as is possible. Fig.2.28 shows the weight-changing circuit proposed in [51–53]. The capacitor C_w stores a weight voltage W, and is connected to one input of the differential pair, M1-M2. The differential pair is biased with a current I_b, while transistors M5-M8 enforce a drain current in M3 of $I_b/2$. The two input voltages of the differential pair, W and W', are thus forced to be equivalent. With a finite current flowing through the resistors *Rhigh* and *Rlow*, the two voltages, W+ and W-, are derived from W' and are slightly higher and lower than W', respectively. W+ and W- are then used to charge and discharge C_w through switches M17 and M18. To control the size of a weight change at each training step, a pulse generator is proposed in [52] for generating variable-width pulses to control the time during which M17 or M18 is switched on. Therefore, the pulse width determines the voltage change at each update, corresponding to the stepsize η_w in Eq.(2.44) [52].

It is notable that the weight-changing circuit proposed in [51–53] does not allow the weight voltage to change over the full range of its power supply. If the weight voltage W is close to 0V (the ground voltage level), the drain current of M1 will become zero and the differential pair will subsequently be disabled. As a result, the equivalence between W and W' no longer holds.

The measurement results in [52] shows that the weight voltage has to be limited to $[2, 5]$V, while the weight-changing circuit operates with a supply voltage of 5V. It is thus important to consider an improved circuit or normalising scheme if the limited ranges of parameters turn out to restrict a model's representational ability.

2.9.3 Discussion

Both the MCD training circuit proposed in [52, 53] and the training circuit for the Boltzmann Machine perform "directional" training, instead of "precise-value" training, to ease the design of training circuits. This suggests that simplifications of training rules are necessary, even if a training rule already displays a hardware-friendly form. By employing an accumulator and an analogue weight-changing circuit, the MCD training circuit further provides a sophisticated tuning of parameters in VLSI. This advancement enables a model not only to capture details of continuous data, but also to minimise the training errors introduced by a simplified training rule. Therefore, the MCD training circuit suggests critical techniques for training a probabilistic generative model, on-chip, to model continuous-valued data.

However, the MCD training circuit faces several limitations. Under the compromise with hardware area, the accumulator computes an average over only a limited number of training data. In addition, the weight-changing circuit in Fig.2.28 provides a limited voltage range for its weight. These limitations inevitably impacts upon the performance of a model in VLSI. To ensure an acceptable performance, these limitations should either be taken into account in simulation or be removed by an improved circuit design.

2.10 Summary for probabilistic neural computation in VLSI

The VLSI implementation of the Boltzmann Machine demonstrates that the use of artificially-generated noise is an efficient way to implement probabilistic computation in VLSI. The development of the Diffusion Network further encourages the exploration of continuous-valued probabilistic dynamics induced by Gaussian noise. As this research intends to identify a continuous probabilistic model from the family of the Recurrent Neural Network, the VLSI implementation of the identified model could draw upon VLSI neurons with noise-induced continuous-stochastic behaviour. The block diagram in Fig.2.29 illustrates how such a continuous-stochastic VLSI neuron can form. The block diagram is derived simply by removing the output comparator in Fig.2.18. The noise input renders the neuron's output continuous-valued and probabilistic. The VLSI circuits of the individual blocks have been suggested in the VLSI implementation of the Boltzmann Machine. However, the grey colours in Fig.2.29 highlight the components to be improved, in order to enhance continuous-valued computation in VLSI. Improved circuits for these blocks will be suggested in Ch.6.

The noise-induced, continuous-valued probabilistic behaviour VLSI remains little explored. The stochastic arithmetic computation in [12–18] bases on binary-valued probabilistic behaviour, while the VLSI implementation of the Bayesian Networks [8–11] or the Support Vector Machine [61–63] merely uses VLSI circuits to calculate conditional probabilities and vector products precisely. As continuous probabilistic behaviour could enhance the robustness of VLSI circuits, computation based on VLSI neurons with noise-induced stochasticity is what this research aims to explore.

The MCD training circuit discussed in Sec.2.9 suggests the VLSI techniques essential for training a probabilistic model, on-chip, to model continuous data. This work also indicates that a training rule not only requires further simplifications to facilitates its hardware implementa-

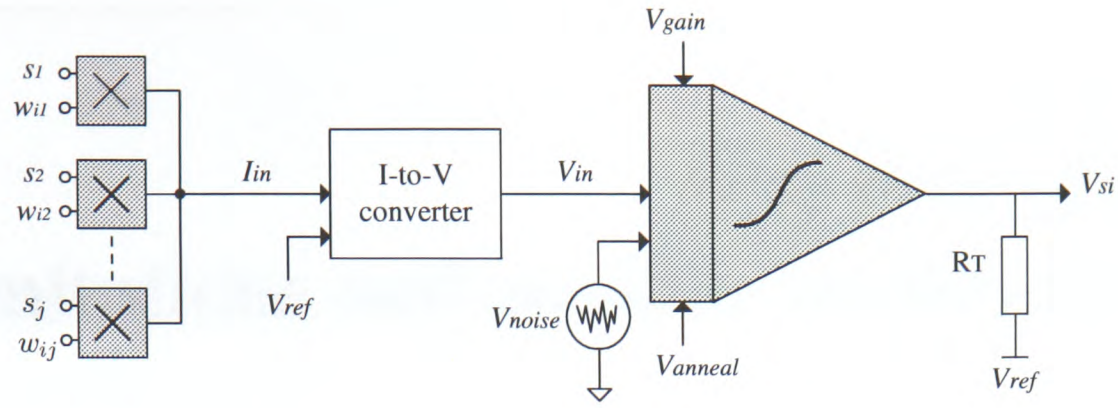


Figure 2.29: *The block diagram of the continuous stochastic neuron this research aims to realise in VLSI.*

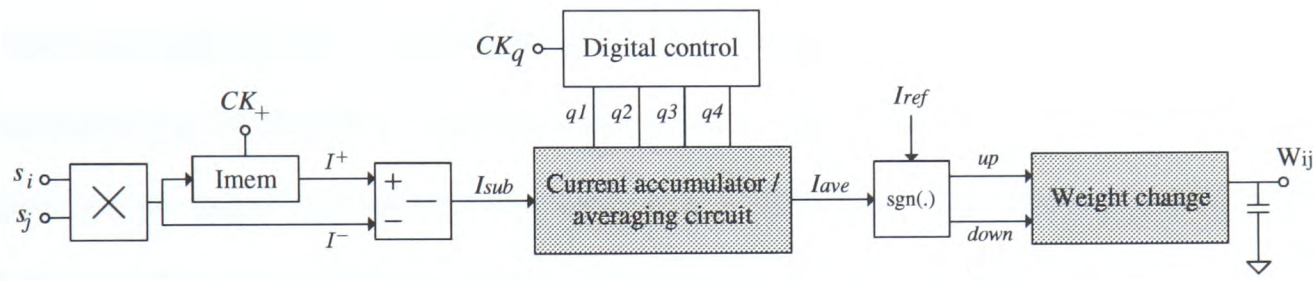


Figure 2.30: *The block diagram of the MCD training circuit this research aims to improve.*

tion, but also faces some hardware limitations. Such simplifications and limitations inevitably introduce additional training errors. It is important to ensure that the training errors do not degrade the performance of a model unacceptably, before implementing the training rule in VLSI. Fig.2.30 shows architecture of a MCD training circuit. The grey colours in Fig.2.30 highlight the blocks that either introduce training errors or require an improved design. Improved circuits for these blocks will be suggested in Ch.7.

Chapter 3
The PoE/RBM : limitations and possible modifications

The PoE/RBM has been shown to be able to model continuous data, with a simple and hardware-amenable training rule [24, 25]. The PoE/RBM therefore suggests a potential probabilistic model for underpinning an embedded intelligent system. However, it is of concern that the PoE/RBM may be limited to represent only specific forms of continuous data, owing to its intrinsic binary nature and deterministic behaviour in visible neurons. The rate-code Restricted Boltzmann Machine (RBMrate) inherits the architecture of the PoE/RBM [64], whilst employing *discrete-valued stochastic* neurons. The RBMrate neurons adopt discrete values, and thus provide greater modelling flexibility. In an attempt to identify a continuous probabilistic model that is both useful and hardware-amenable, this chapter examines the capabilities of both models with various continuous data. Two-dimensional artificial data will be used in many experiments, in order to visualise the performance of various models. The limitations of the PoE/RBM, together with the possible modifications suggested by the RBMrate, will subsequently be identified. These results provide the basis for developing the continuous probabilistic model in chapter 4.

3.1 The limitations of the PoE/RBM

Fig.3.1(a) shows a PoE/RBM with two visible and four hidden neurons. The connections between neurons are bidirectional and symmetric. The vector $\mathbf{w}^{(i)}$ denotes the weight vector of hidden neuron h_i, containing the connections between h_i and the two visible neurons. The two

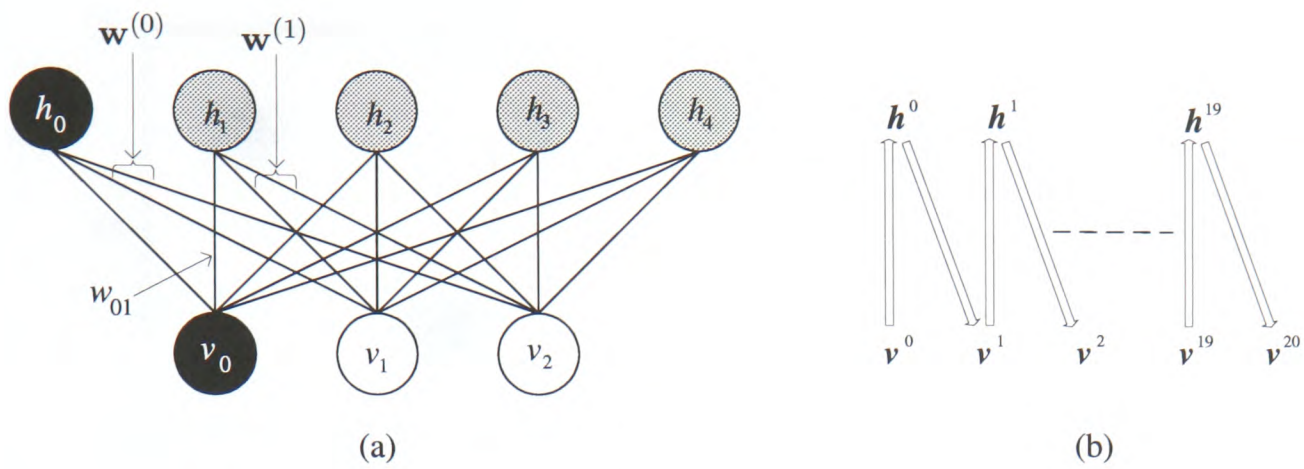


Figure 3.1: *(a)A PoE/RBM with two visible and four hidden neurons, as well as two bias units. (b)The diagram illustrating the 20-step Gibbs sampling from random initial states* $\{\mathbf{v}^0\}$. *The sampled visible states at the 20th-step* $\{\mathbf{v}^{20}\}$ *are used to approximate the equilibrium reconstruction.*

black circles represent *bias units* whose outputs are 1, and the connection from a bias unit to a neuron is thus the *threshold* of the neuron. For example, the connection w_{01} in Fig.3.1(a) is equivalent to a threshold input of $-w_{01}$ for the hidden neuron h_1.

The MCD training rule in Eq.(2.35) can be re-written as the following

$$\Delta w_{ij} = \eta_w \left(\langle pv_i \cdot ph_j \rangle_0 - \langle pv_i \cdot ph_j \rangle_1 \right) \tag{3.1}$$

where η_w is a constant that determines the *learning rate* during training, for all connection weights $\{w_{ij}\}$. The following subsection first demonstrates how the PoE/RBM models simple two-dimensional data. Sec.3.1.2 and Sec.3.1.3 will then complicate the distribution of training data, in order to examine the modelling capability of the PoE/RBM.

3.1.1 Modelling continuous-valued data

To demonstrate an ability to model continuous-valued data, a PoE/RBM, with two visible and four hidden neurons (Fig.3.1(a)), was trained to model the training data in Fig.3.2(a). The

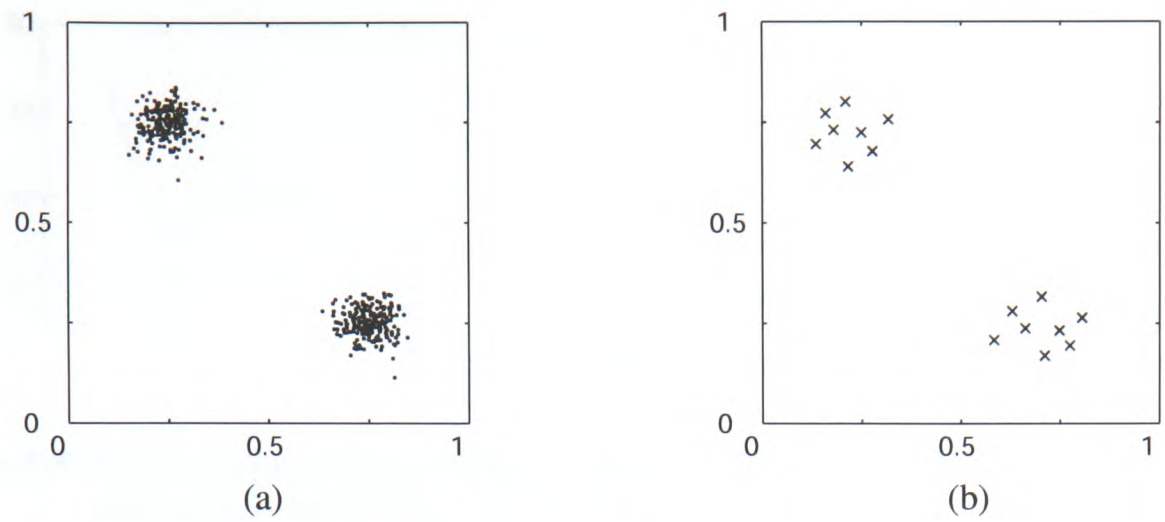


Figure 3.2: *(a)The training data containing two clusters of 200 data points in Gaussian distribution. (b)The 20-step reconstruction generated by the trained PoE/RBM from 400 random initial data*

training data contain two well-separated clusters of 200 data points in Gaussian distributions.

The PoE/RBM was trained for 1000 epochs[1] with $\eta_w = 1$. To visualise the performance of the trained model, an "approximate equilibrium reconstruction" was obtained by Gibbs sampling from random initial visible states $\{\mathbf{v}^0\}$ for 20 steps. Fig.3.1(b) illustrates the 20-step Gibbs sampling process. With each initial visible state $\mathbf{v}^0$, hidden and visible neurons are sampled alternately for 20 iterations, until the 20-th step reconstructed state $\mathbf{v}^{20}$ is obtained. To evaluate how well the 20-step reconstruction $\{\mathbf{v}^{20}\}$ approximates equilibrium reconstruction, the Euclidean distance between n-th and $(n-1)$-th visible states, $\|\mathbf{v}^n - \mathbf{v}^{(n-1)}\|$, was calculated during the multi-step Gibbs sampling process. As explained in Ch.2, multi-step Gibbs sampling effectively leads a model towards states with lower energy (i.e. higher probabilities). If $\mathbf{v}^{(n-1)}$ corresponds to a local minimum in the energy landscape, the probability that the model will "travel a long distance" at the next step is relatively small. The Euclidean distance between $\mathbf{v}^n$ and $\mathbf{v}^{(n-1)}$ should therefore become relatively small when multi-step reconstruction approximates equilibrium reconstruction. Fig.3.3 shows the Euclidean distance during 20-step Gibbs sampling, summed over 400 trials with random initial data. Obviously, the sum of Euclidean

[1] 1 epoch refers to updating parameters once according to Eq.(3.1)

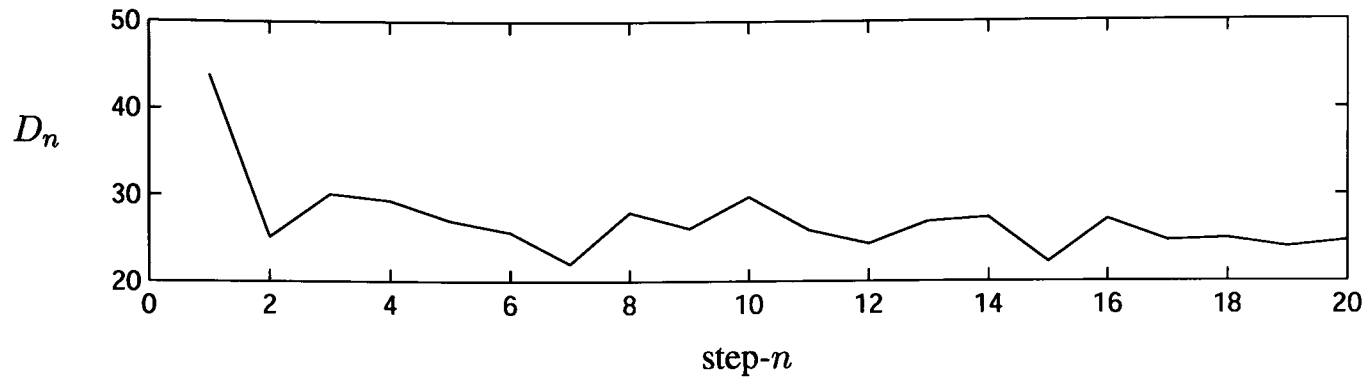


Figure 3.3: *The Euclidean distance during 20 steps of Gibbs sampling, summed over 400 trials with random initial data.* $D_n = \sum_{\mathbf{v}} \|\mathbf{v}^n - \mathbf{v}^{(n-1)}\|$.

distance approaches a relative minimum value in 10 steps, indicating that 20-step reconstruction is a reasonable approximation for equilibrium reconstruction in this experiment.

Fig.3.2(b) shows the 20-step reconstruction by the trained PoE/RBM, from 400 random initial data. The distribution of the reconstructed data evidently approximates that of the training data. To visualise how the trained model regenerates the data distribution, Fig.3.4 displays the weight vectors of hidden bias unit and hidden neurons, by projecting $\{\mathbf{w}^{(i)}\}$ into the state space of visible neurons. The projection means passing $\{\mathbf{w}^{(i)}\}$ through the *sigmoid* function $\sigma(\cdot)$ in Eq.(2.7), i.e. $\mathbf{r}^{(0)} = \sigma(\mathbf{w}^{(i)})$ in Fig.3.4. As Eq.(2.33) indicates that the post-sigmoid probabilities of visible neurons are treated directly as the continuous-valued outputs of visible neurons, the projected vectors reveal the effects of hidden neurons on the distribution of reconstructed data.

For example, if all hidden neurons are off (outputs equal to 0) at a particular step of sampling visible neurons, $\mathbf{w}^{(0)}$ is the only input to visible neurons, resulting in the sampled visible state to be $\sigma(\mathbf{w}^{(0)})$. The reconstructed point at this particular step is therefore the star in Fig.3.4(a). In other words, the "permanently-on" hidden bias unit creates the star as the "default reconstructed point" when all hidden neurons are off. The turn-on of hidden neuron i, on the other hand, "shifts" the reconstructed point from the star to a new point, with $\mathbf{r}^{(i)}$ defining

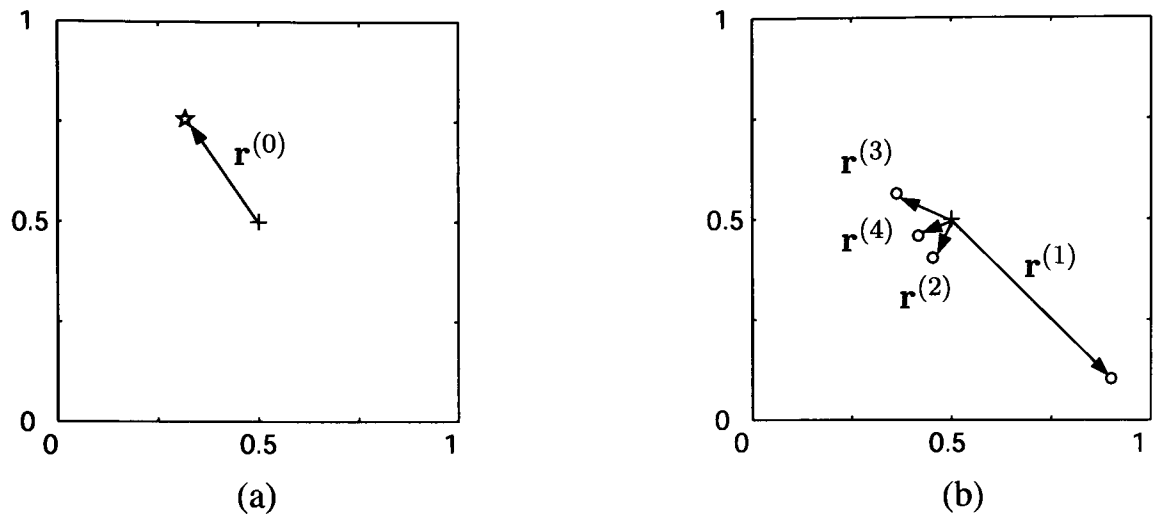


Figure 3.4: *(a)The projection of* $\mathbf{w}^{(0)}$ *in the state space of visible neurons. (b)The projection of the weight vectors* $\{\mathbf{w}^{(i)}\}$ *of the four hidden neurons in the state space of visible neurons.*

the "shifting vector". Fig.3.4(a) indicates that the default reconstructed point is close to the position of the top-left cluster, while Fig.3.4(b) shows that the hidden neuron h_1 encodes the position of the bottom-right cluster relative to $\mathbf{r}^{(0)}$. All other hidden neurons, h_2-h_4, then encode the variation of data in different directions. Therefore, if h_1 is off, reconstructed points will distribute around the top-left cluster, while if h_1 is on, reconstructed points will distribute around the bottom-right cluster.

This simple experiment demonstrates how the PoE/RBM models continuous data, but also exposes the model's first limitation. Fig.3.2(b) shows that the reconstructed data comprise 16 discrete data points. A PoE/RBM with N hidden neurons can reconstruct only 2^N discrete data points, because of the binary nature of its hidden neurons. The reconstruction in Fig.3.2(b) thus contains only $16(= 2^4)$ points, despite Gibbs sampling from 400 random initial data. Although it is possible to ease this limitation by recruiting extra hidden neurons, binary hidden neurons introduce other limitations, as described in the next subsection.

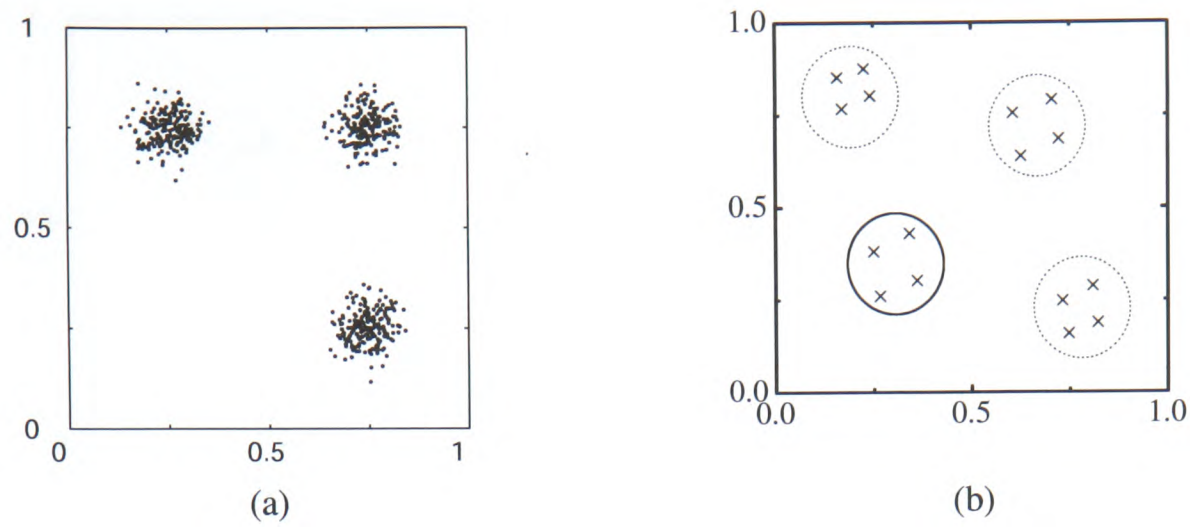


Figure 3.5: *(a)The training data containing three clusters of 200 data points (b)The 20-step reconstruction generated by the trained PoE/RBM from 400 random initial data*

3.1.2 Training data with a non-symmetric distribution

Consider the training data in Fig.3.5(a). The training data contain three clusters of 200 Gaussian-distributed data points. The PoE/RBM with two visible and four hidden neurons was trained on this data for 2000 epochs with $\eta_w = 1.5$. After training, the PoE/RBM generated the 20-step reconstruction shown in Fig.3.5(b). The 12 data points circled by the dashed lines correspond to the three clusters in the training data. Although the centers of the dashed circles do not match the centers of the three clusters faithfully, this result indicates that the PoE/RBM has captured the distribution of the three clusters. The PoE/RBM, however, reconstructed extra data points around the bottom-left region, circled by the solid line. The occurrence of these extra data points is attributed to the binary nature of hidden neurons, as explained in the following.

Fig.3.6 displays the projection of the trained $\{\mathbf{w}^{(i)}\}$ in the state space of visible neurons. As the bias vector $\mathbf{w}^{(0)}$ encodes the position of the top-left cluster, $\mathbf{w}^{(2)}$ and $\mathbf{w}^{(4)}$ shift the reconstructed points to the positions of the other two clusters. If h_2 is on and h_4 off, the reconstructed point will distribute around the top-right cluster. If both h_2 and h_4 are on, the reconstructed points will distribute around the bottom-right cluster. However, the probability for h_2 being off and h_4 being on is non-zero, resulting in reconstructed points around the

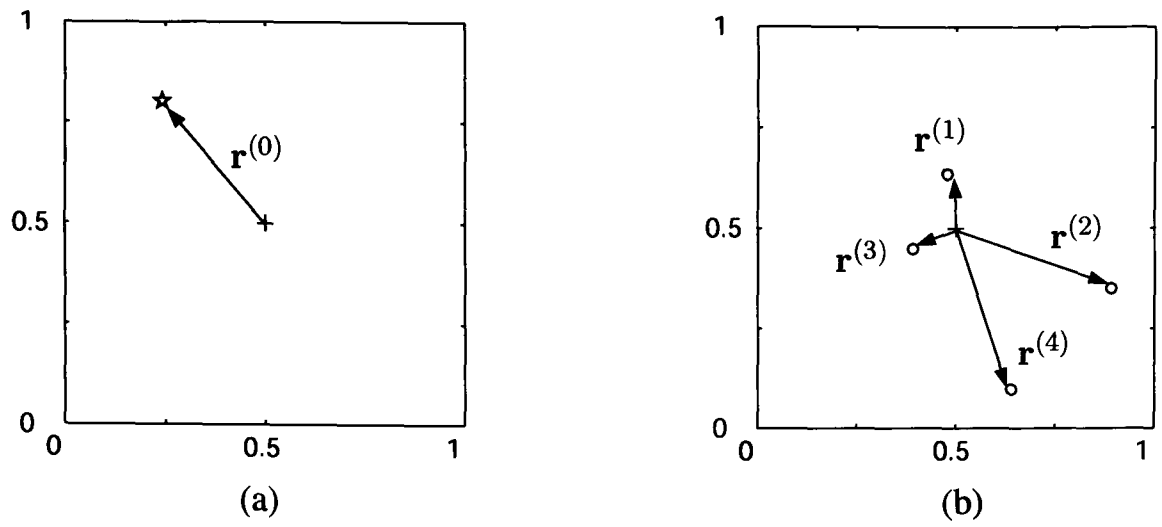


Figure 3.6: *The projection of* $\mathbf{w}^{(0)}$ *in the visible space. (b)The projection of* $\{\mathbf{w}^{(i)}\}$ *of the four hidden neurons in the visible space.*

bottom-left region. The occurrence of these extra data points is therefore unavoidable. More specifically, the binary nature of hidden neurons forces the PoE/RBM to reconstruct data points with a "symmetric" distribution.

Although the PoE/RBM reconstructs symmetrically-distributed data points, it is important to examine whether the PoE/RBM at least assigns lower probability to the extra data points. Fig.3.7(a) shows the statistical density of the 16 reconstructed points, estimated by sampling 1000 reconstructed data from the trained PoE/RBM. The number next to each point gives the occurring probability of the point, in the unit of percentage (%). Fig.3.7(b) plots the statistical density over the visible-state space. Clearly, the PoE/RBM assigns lower probability to the space around the bottom-left region. However, the statistical density differs non-negligibly from the real probability density of the training data. To inspect whether the probability distribution learnt by the PoE/RBM indeed matches that of training data, more complicated data are used in the next subsection.

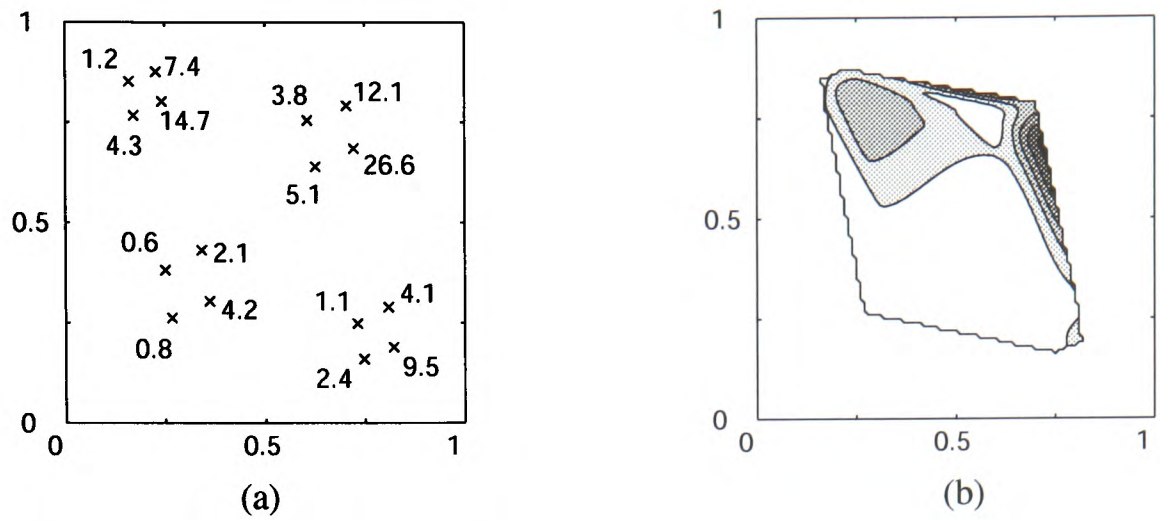


Figure 3.7: *(a)The statistical density of the reconstructed data points in Fig.3.5(b). (b)The statistical density over the visible-state space. The darker colour indicates a higher probability.*

3.1.3 The probability distribution of reconstructed data

To examine the probability distribution modelled by a PoE/RBM, the $\tanh(\cdot)$ function was used as the output function for visible neurons in this experiment. The $tanh(\cdot)$ function is defined as

$$v_i = \tanh\left(\frac{x_i}{2}\right) = \frac{1-\exp(-x_i)}{1+\exp(-x_i)} \quad , \qquad \text{where} \quad x_i = \sum_j w_{ij} h_j \tag{3.2}$$

The $\tanh(\cdot)$ function retains the same sigmoidal nonlinearity as the *logistic* function in Eq.(2.7), but scales the output range of visible neurons to $[-1, 1]$, instead of $[0, 1]$. This change enables the projections of weight vectors $\{\mathbf{w}^{(i)}\}$ to have the same origin as data vectors in the visible-state space.

Without this change, the projections of $\{\mathbf{w}^{(i)}\}$ have origin at $(0.5, 0.5)$, as shown in Fig.3.4 and Fig.3.6, while the origin of data vectors is at $(0, 0)$. The difference between the two origins implicitly prevents a modelled probability distribution from agreeing with the probability

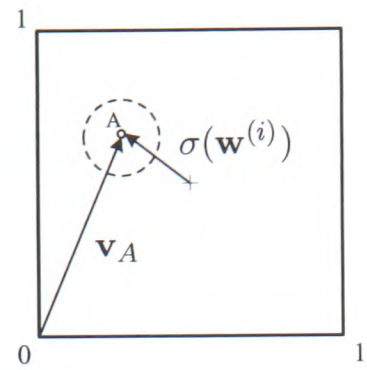


Figure 3.8: *The difference between the origins of $\sigma(\mathbf{w}^{(i)})$ and the data vector $\mathbf{v}_A$ impedes the PoE/RBM from assigning high probability to the data point A*

distribution of training data. For example, let the weight vector of one hidden neuron be projected as the vector $\sigma(\mathbf{w}^{(i)})$ in Fig.3.8. The turn-on of the hidden neuron enables the model to reconstruct data points around the dashline-circled region, provided $\mathbf{w}^{(0)} = 0$. However, the vector of data point A, for instance, is almost perpendicular to $\sigma(\mathbf{w}^{(i)})$ and thus to $\mathbf{w}^{(i)}$. This consequently prevents the model from assigning high probability to data points around the dashline-circled region, as indicated by Eq.(2.26). The $tanh(\cdot)$ function was therefore used as the output function of visible neurons, to rescale the value range of visible neurons without altering a model's behaviour.

A PoE/RBM with two visible and six hidden neurons was trained to model the training data in Fig.3.9(a). The training data contain two clusters of 200 data points in Gaussian distributions. One cluster is elliptical and the other circular. After 4000 training epochs with $\eta_w = 1.5$, the PoE/RBM generated the 20-step reconstruction as shown in Fig.3.9(b). The reconstruction is inevitably symmetric, but it is of more interest to examine whether the trained model at least defines a correct probability distribution over the visible-state space. The probability distribution was estimated by sampling 1000 reconstructed data from the trained model, computing the statistical density of the reconstructed points, and then plotting the density over the visible-state space. Fig.3.10 shows the estimated statistical density over the visible-state space, with

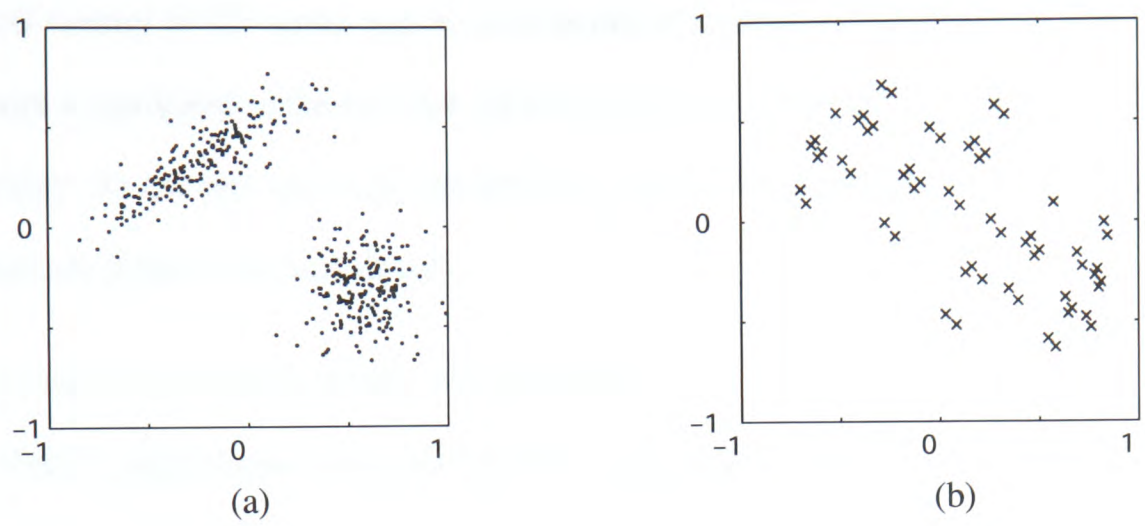


Figure 3.9: *(a)The training data containing two clusters of 200 data points in Gaussian distribution. One Gaussian is elliptical, and the other circular. (b)The 20-step reconstruction generated by the trained PoE/RBM from 400 random initial data*

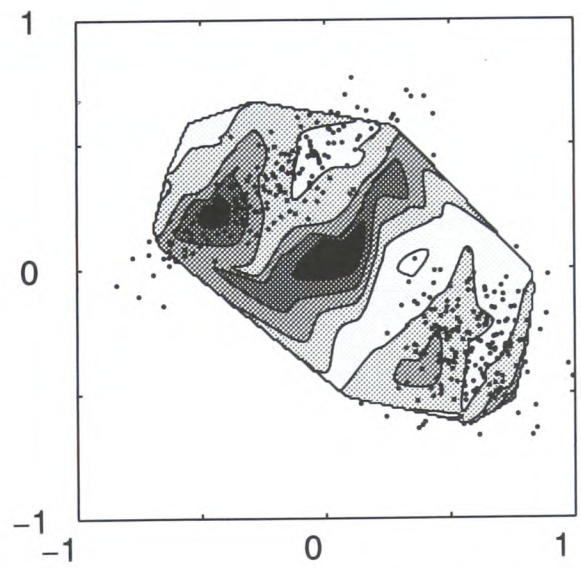


Figure 3.10: *The estimated statistical density assigned by the PoE/RBM when modelling trained on the training data in Fig.3.9(a). The darker colour indicates a higher probability, and the training data points are superimposed on the contour plot.*

the training data points superimposed on the distribution. Fig.3.10 reveals that the statistical density differs significantly from the real distribution of the training data. The relatively high statistical density in the center region corresponds to an area with no training data. This disagreement is attributed to the fact that the bias vector $\mathbf{w}^{(0)}$ pointed to the center region in this experiment. As the bias unit is permanently-on, the region pointed by the bias vector always has relatively higher statistical density.

Consider the probability density function defined by a PoE/RBM with weights $\mathbf{W} = \{w_{ij}\}$. Since Hinton approximates the post-sigmoid probabilities of visible neurons directly as the continuous-valued outputs of visible neurons, the probability density function is of the same form as Eq.(2.26), and can be rewritten as

$$Pr(\mathbf{v} \mid \mathbf{W}) = \frac{1}{Z}\Pi_i \left(1 + \exp^{\mathbf{w}^{(i)}.\mathbf{v}}\right) \tag{3.3}$$

The only difference from Eq.(2.26) is that $\mathbf{v}$ in Eq.(3.3) is continuous-valued for the PoE/RBM. Under Hinton's approximation, the component densities introduced by individual experts, $(1 + \exp^{\mathbf{w}^{(i)}.\mathbf{v}})$, always have maxima or minima on the boundaries of a visible-state space, because the exponential function is a monotonically increasing function. The maxima of $Pr(\mathbf{v} \mid \mathbf{W})$ thus only occur on the boundaries of the visible-state space, and the bias unit of the model merely introduces a relatively high statistical density, which does not necessarily correspond to the distribution of training data. This limits the ability of the PoE/RBM in modelling continuous-valued probabilistic distributions faithfully. Therefore, although Hinton's approximation enables the PoE/RBM to model continuous data, continuous-deterministic visible neurons offer limited flexibility in modelling continuous-valued probability distributions.

3.1.4 Discussion

The limitations of the PoE/RBM are summarised as follows

- The number of distinctive reconstructed data points is limited by the number of hidden neurons.
- The distribution of reconstructed data is always symmetric.
- The inability to model a continuous-valued probability distribution faithfully.

The first two limitations are due to the binary nature of hidden neurons, while the last is due to the deterministic approximation applied to visible neurons. Hinton used the PoE/RBM to classify hand-written digits without suffering from these limitations because one PoE/RBM only modelled one hand-written digit in his experiments [24]. A test image was classified by computing the probabilities under a set of models trained on different digits, and the model assigning highest probability decides its category. Let each image correspond to a data point in a high-dimensional space. Since the variations among the images of a single digit scarcely introduce a clear separation between their corresponding data points, the training task in Hinton's experiments is more analogous to training one PoE/RBM to model only one cluster of data. In Murray's experiments with real heartbeat data in [25], although one PoE/RBM was trained to model both normal and abnormal heartbeats, the trained PoE/RBM reconstructed only normal heartbeats. This indicates that the trained PoE/RBM effectively modelled only one category of data (normal heartbeat), and the other category (abnormal heartbeat) was detected by the fact that the PoE/RBM assigns low probabilities to data which does not belong to its modelled category [25]. The ability to model a non-symmetric data distribution or complex probability distribution is therefore irrelevant in these experiments. However, multiple PoE/RBM models are impractical and expensive for an intelligent system, especially when the number of data

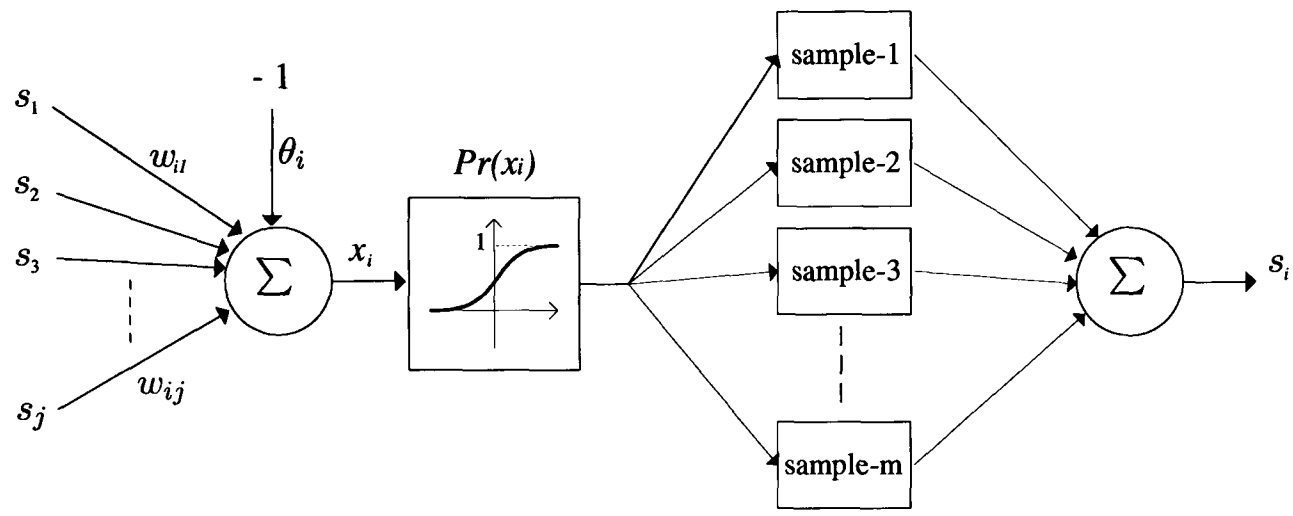


Figure 3.11: *The neuron of the rate-coded Restricted Boltzmann Machine. The m binary samples is summed up to yield a discrete-valued probabilistic output*

categories is unknown. So the PoE/RBM at least requires some modification to suit the applications of our interest. The rate-coded Restricted Boltzmann Machine suggests one possible modification, as discussed in the next section.

3.2 The rate-coded Restricted Boltzmann Machine

As a modified model of the PoE/RBM, the "rate-coded" Restricted Boltzmann Machine (abbreviated as RBMrate in the following) [64] lifts some of the limitations by estimating the "firing rate" of binary-stochastic neurons. This estimate is achieved by sampling each binary-stochastic neuron for m times, representing continuous values by the number of active samples. The neurons of the RBMrate thus have *discrete-valued* outputs of $\{1, 2, 3, \ldots, m\}$, as shown in Fig.3.11.

Employing discrete-valued stochastic neurons in a hidden layer, the RBMrate is able to reconstruct a considerable number of distinctive data points with non-symmetric distribution. Using discrete-valued stochastic neurons in a visible layer, on the other hand, allows the probability maxima to be inside a continuous space. This is because the repetitive-sampling action in

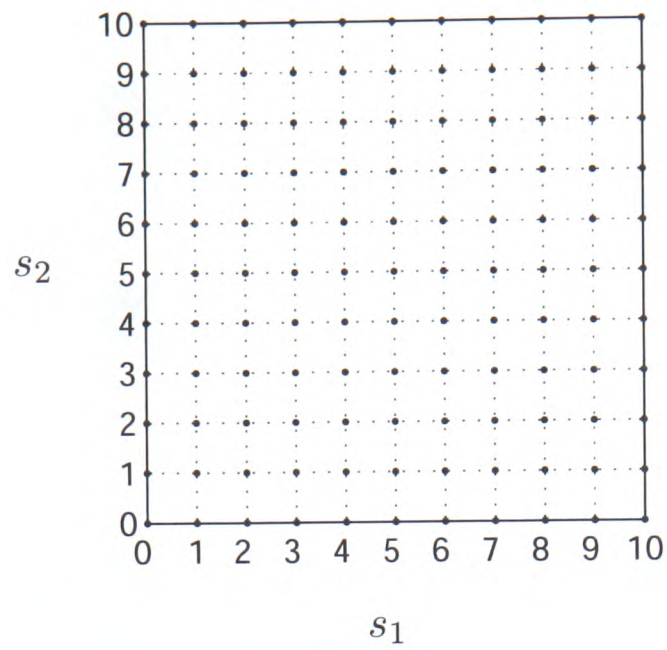


Figure 3.12: *The states of an RBMrate with two neurons, s_1 and s_2, include all the grid points in the two-dimensional space above, assuming $m = 10$ for each neuron.*

Fig.3.11 is equivalent to replicating the binary-stochastic neuron in Fig.2.7(a) into m copies. In other words, one RBMrate neuron with m discrete values is equivalent to m binary neurons in a Restricted Boltzmann Machine (RBM). Each discrete-valued state of an RBMrate therefore corresponds to a binary state in the hyper-cubic state space of a particular RBM. For example, an RBMrate with two neurons and $m = 10$ has the two-dimensional discrete-valued states in Fig.3.12. These states can be viewed as spreading the binary states of a 20-neuron RBM onto the two-dimensional plane. Since any binary state of a RBM can correspond to an energy minimum, the energy minima (or the probability maxima) of the RBMrate can be "inside" the plane of continuous space.

The RBMrate retains the same network structure as the PoE/RBM, and preserves the MCD training rule in Eq.(2.31) without loss of rigour [64]. Let v_i and h_j represent the discrete-valued outputs of visible neuron i and hidden neuron j, respectively. The training rule for the RBMrate is written as

$$\Delta w_{ij} = \eta_w \left(\langle v_i \cdot h_j \rangle_0 - \langle v_i \cdot h_j \rangle_1 \right) \tag{3.4}$$

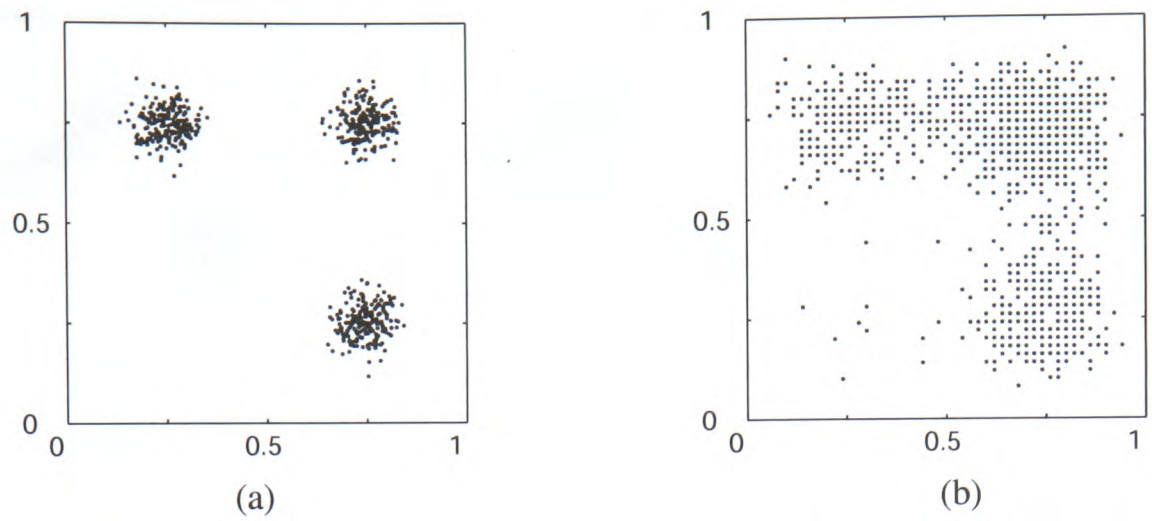


Figure 3.13: *(a)The training data containing three clusters of 200 data points (b)The data points reconstructed by the trained RBMrate after 20 steps of Gibbs sampling, from 400 random initial data.*

where η_w is the learning rate. The following subsections shows the RBMrate's performance on modelling the same training data used in Sec.3.1.2 and Sec.3.1.3, for comparison with the PoE/RBM.

3.2.1 Training data with a non-symmetric distribution

An RBMrate with two visible and four hidden neurons was trained to model the three-cluster training data in Fig.3.13(a). The number of sampling times, m, for neurons was selected empirically. m for visible neurons was selected to provide the RBMrate with enough discrete-valued levels between $[0, 1]$, in order to regenerate distributions with acceptable discretising artefacts. m for hidden neurons was then selected to provide the RBMrate with enough modelling flexibility to lift the limitations faced by the PoE/RBM. m was set to 50 for visible neurons and to 10 for hidden neurons. The discrete-valued output of each neuron was normalised with respect to m, such that the output values $\{s_i\}$ lie in $[0, 1]$. After 2000 training epochs with $\eta_w = 0.015$, the RBMrate reconstructed data as shown in Fig.3.13(b). Note that the approximate equilibrium reconstruction was also obtained by Gibbs sampling from 400 random initial data for 20

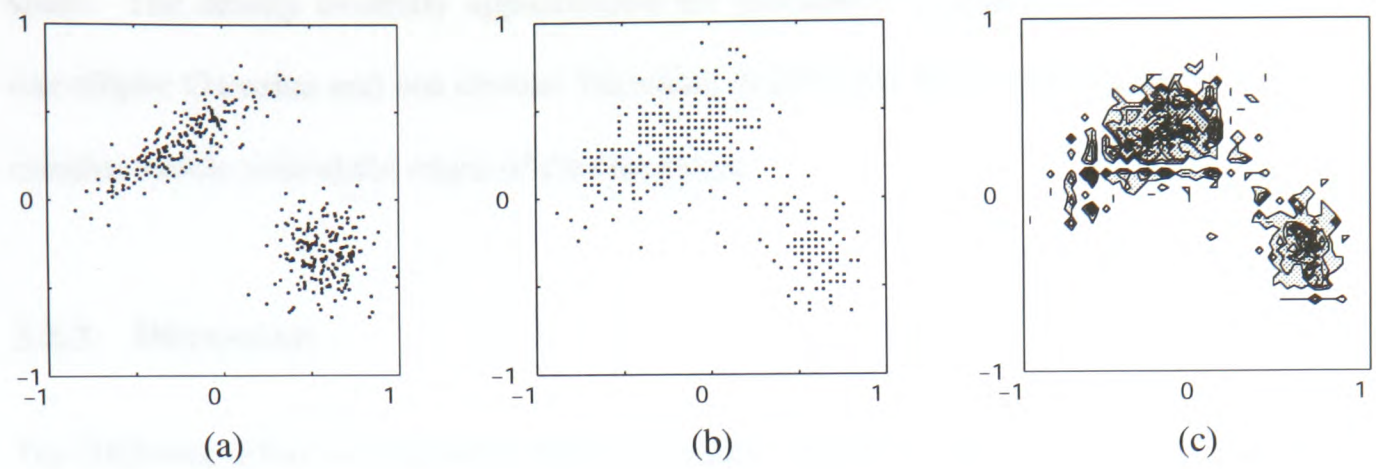


Figure 3.14: *(a)The training data containing two clusters of 200 data points in Gaussian distribution. One Gaussian is elliptical, and the other circular. (b)The 20-step reconstruction generated by the trained RBMrate from 400 random initial data (c)The statistical density over the visible-state space assigned by the RBMrate in this experiment. The darker colour indicates a higher probability.*

steps. Fig.3.13 reveals that the RBMrate is able to reconstruct data points with a non-symmetric distribution, although the separation between clusters is not as clear as that of the training data Fig.3.5(a). The discrete-valued nature of neurons is also revealed by the "graininess" in the reconstructed data.

3.2.2 The probability distribution of reconstructed data

To examine the RBMrate's capability to model continuous-valued probability distribution, an RBMrate with two visible and six hidden neurons was trained to model the training data in Fig.3.14(a). The number of sampling times was set to 50 for visible neurons and to 10 for hidden neurons. After 4000 training epochs with $\eta_w = 0.04$, the RBMrate generated the 20-step reconstruction as shown in Fig.3.14(b). The RBMrate's ability to regenerate data with non-symmetric distribution is demonstrated once more in this experiment. To estimate the statistical density of the reconstructed data, the same way used in Sec.3.1.3 was adopted. 1000 reconstructed data were sampled and the occurring probability of each discrete-valued state was then calculated. Fig.3.14(c) shows the estimated statistical density over the visible-state

space. The density evidently approximates the probability distribution of the training data, one elliptic Gaussian and one circular Gaussian, despite that the discretising effect of neurons remains visible around the edges of the Gaussians.

3.2.3 Discussion

The RBMrate offers an improved ability to model continuous data, as was also demonstrated with continuous image data in [64]. However, the repetitive sampling of the RBMrate is inconvenient and will exacerbate noise in the power supplies of VLSI implementation. Repeated sampling actions not only retard the speed of simulating the RBMrate in both software and hardware, but also place VLSI circuits in danger of synchronisation [65]. In addition, the increase of sampling times is essential for modelling complex continuous data. The RBMrate is therefore unsuitable for VLSI implementation.

In spite of this drawback, the discrete-valued stochastic neuron of the RBMrate suggests an alternative way of introducing stochastic behaviour. Note that repetitively sampling according to a fixed probability p_i actually gives rise to a *Bernoulli* random variable with an expectation value of p_i [66] [2]. An RBMrate neuron is thus able to be transformed into the neuron in Fig.3.15, where the addition of a *Bernoulli noise* with zero mean is substituted for repetitive sampling. This perspective attracted our attention to the continuous-stochastic neuron proposed by Frey [67], as shown in Fig.3.16. With the Gaussian noise input, this neuron has a continuous-valued probabilistic output. It is notable that Alspector also utilised Gaussian noise to implement stochastic neurons in VLSI [50], as discussed in Sec.2.8. Moreover, Alspector proposed an efficient implementation of multiple channels of noise in VLSI [21]. The newly-interpreted

[2] Let s be a binary random variable with probabilities $Pr(s=1)=p$ and $Pr(s=0)=1-p$, and $0 \leq p \leq 1$. s is called a Bernoulli random variable. Sampling s for n times gives the probability density function as

$$Pr(i) = \left(\begin{array}{c} n \\ i \end{array} \right) p^i (1-p)^{n-i} \tag{3.5}$$

where i is the number of counts for $s=1$, and the expectation value of i equals to p.

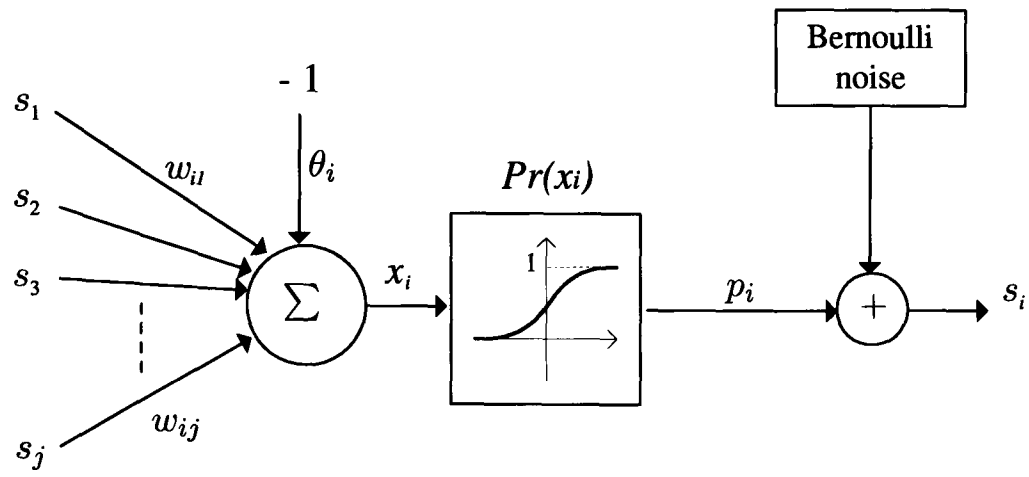


Figure 3.15: *The neuron that possesses the equivalent probabilistic behaviour as the neuron of the RBMrate.*

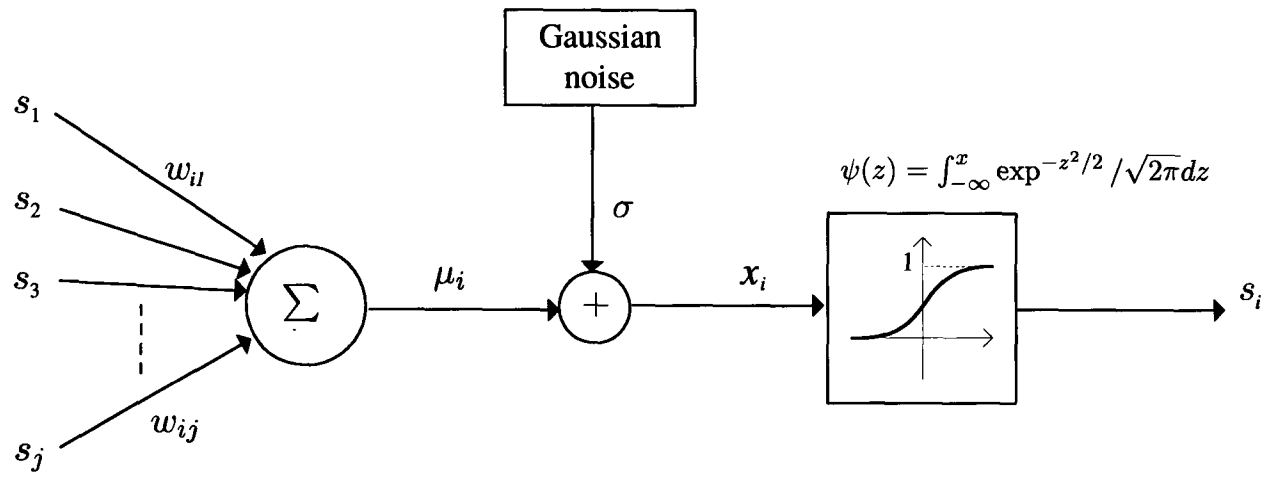


Figure 3.16: *The continuous-stochastic neuron proposed by Frey. The nonlinear function $\psi(z)$ is called the cumulative Gaussian function.*

RBMrate neuron in Fig.3.15, together with the works discussed above, therefore suggest that continuous probabilistic computation in VLSI could be based on continuous stochastic neurons with noise-induced stochasticity.

3.3 Summary

The PoE/RBM has limited ability to model continuous-valued data because the PoE/RBM is actually composed of binary-stochastic neurons and continuous-deterministic neurons. The RBMrate provides an improved modelling ability, whereas its discrete-valued stochastic neu-

rons require repetitive Gibbs sampling, making the RBMrate non-ideal for hardware implementation. Nevertheless, the simulation results with RBMrate demonstrates that discrete-valued stochastic neurons offer greater modelling flexibility than do binary-stochastic neurons. It is expected that continuous-valued stochastic neurons will further enrich the representational power and stochastic behaviour of a model. The RBMrate neuron in Fig.3.15 hints at an alternative way of introducing continuous stochastic neurons. Following this lead, a continuous-valued probabilistic model has been developed in this research. The development of this model is described in next chapter.

Chapter 4
A continuous-valued probabilistic model with a hardware-amenable training algorithm

The objective in this chapter is to develop a continuous probabilistic model that is both useful in modelling continuous data and amenable to hardware implementation. The results in Ch.3 suggest that a continuous stochastic neuron can be introduced by adding Gaussian noise to the input of a neuron [67]. Following this lead, the *Continuous Restricted Boltzmann Machine* [1] (abbreviated as CRBM in the rest of the thesis) [68] is developed by substituting these "noisy neurons" into the restricted architecture of the PoE/RBM. With real continuous stochastic behaviour, the CRBM potentially has richer representational power than both the PoE/RBM and the RBMrate. The relationship between the CRBM and the Diffusion Network, furthermore, leads to the derivation of a hardware-amenable training algorithm for the CRBM. This chapter describes the development of the CRBM with its hardware-amenable training rules. The capabilities of the CRBM will also be demonstrated and explored with both artificial and real heart-beat data.

4.1 The Continuous Restricted Boltzmann Machine

4.1.1 Continuous stochastic neuron

The *Continuous Restricted Boltzmann Machine* consists of continuous stochastic neurons that incorporate Gaussian input noise, as shown in Fig.4.1. $N_i(0,1)$ represents a Gaussian random variable with zero mean and unit variance. The constant σ and $N_i(0,1)$ thus constitute a noise

[1] The equilibrium of the CRBM only approximates Boltzmann equilibrium, and will be discussed in Sec.4.2.

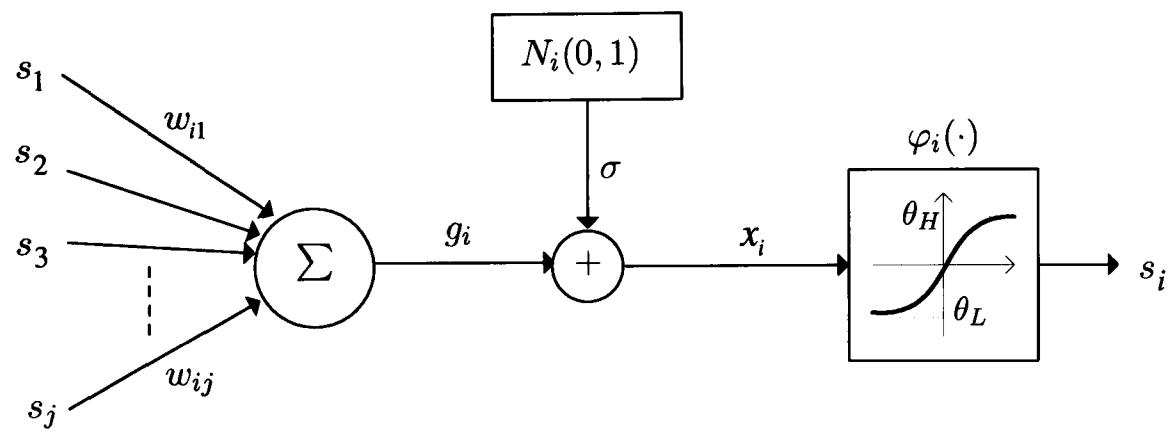


Figure 4.1: *The continuous stochastic neuron employed by the Continuous Restricted Boltzmann Machine*

input component, $n_i = \sigma \cdot N_i(0,1)$, according to the following probability distribution:

$$p(n_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-n_i^2}{2\sigma^2}\right) \tag{4.1}$$

Let s_i denote the output of the neuron, and w_{ij} the connection between neuron i and neuron j. The output of the neuron is

$$s_i = \varphi_i\left(\sum_j w_{ij}s_j + \sigma \cdot N_i(0,1)\right) \tag{4.2}$$

$$\text{with} \quad \varphi_i(x_i) = \theta_L + (\theta_H - \theta_L) \cdot \frac{1}{1+\exp(-a_i x_i)} \tag{4.3}$$

where $\varphi_i(x)$ is a sigmoid function with asymptotes at θ_L and θ_H. Let $g_i = \sum_j w_{ij}s_j$. Fig.4.1 shows that the probability density of x_i is a Gaussian distribution with mean g_i and variance σ^2, denoted as $Pr(x_i) = \mathcal{G}(g_i, \sigma^2)$. The probability density of s_i is thus of the following form

$$Pr(\, s_i \mid g_i, \sigma) = \frac{\mathcal{G}(g_i, \sigma^2)}{\varphi_i'(x_i)} \tag{4.4}$$

$\varphi_i'(x_i)$ denotes the derivative of $\varphi_i(x_i)$ with respect to x_i. Eq.(4.4) implies that a noise input before the sigmoid function enriches a neuron's stochastic behaviour more than does a noise

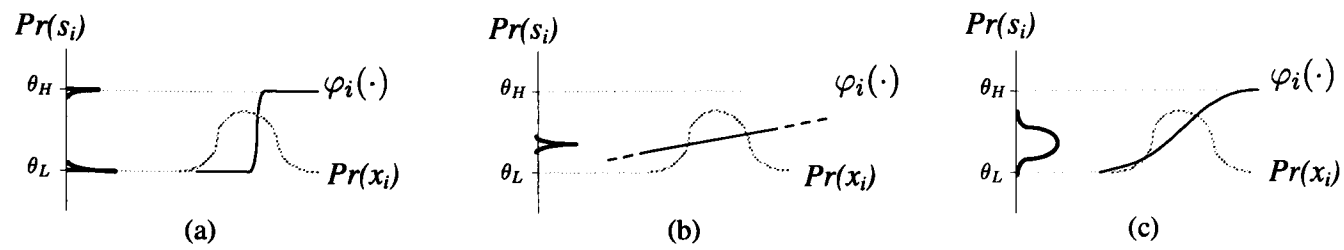


Figure 4.2: *The output probability density of the continuous stochastic neuron in Fig.4.1 can vary with the slope of the sigmoid function (a)near-binary stochastic (b)near-deterministic (c)continuous stochastic*

input after the nonlinear function in Fig.3.15. More specifically, parameter a_i controls the slope of the sigmoid function, and consequently the nature of the neuron's stochastic behaviour, as shown in Fig.4.2. With the pre-sigmoid probability density, $Pr(x_i)$, being a Gaussian, the post-sigmoid probability density, $Pr(s_i)$, varies with the slope of the sigmoid function. Fig.4.2(a) shows that a large value of a_i leads to a sharp sigmoid function, such that most input values of x_i correspond to an output of either θ_H or θ_L. The probability density $Pr(s_i)$ thus contains two sharp peaks at θ_H and θ_L, and the non-zero mean of x_i causes the peak values to be different. This indicates that a large value of a_i introduces an *near-binary stochastic* neuron. In contrast, Fig.4.2(b) shows that a small value of a_i make input noise negligible compared with the linear range of the sigmoid function. $Pr(s_i)$ thus contains a single sharp peak, indicating that a small value of a_i leads to a *near-deterministic* neuron. If the value of a_i renders the sigmoid function *approximately-linear* over the range of the added noise, $Pr(s_i)$ approximates a Gaussian distribution, as shown in Fig.4.2(c). As adapting parameter a_i effectively adapts the influence of input noise on output probability density, parameter a_i is called "noise-control" parameter.

The neuron proposed by Frey also possesses similar stochastic behaviour [67], but differs from the neuron introduced above in several aspects. The nonlinear function of Frey's neuron is the cumulative Gaussian function, as shown in Fig.3.16, rather than the sigmoid function

in Eq.(4.3). Therefore, Frey varies the neuron's stochastic behaviour by tuning the variance σ of noise, rather than adjusting the parameter a_i. Though a large σ also induces a near-binary stochastic neuron, the output of such binary stochastic neuron actually becomes "binary-random" with a probability of 50%, i.e. independent of the deterministic input g_i. The output probability of the near-binary neuron with a large a_i, on the contrary, remains dependent on g_i, as depicted in Fig.4.2(a). Sec.4.5 will show that a hidden neuron with a large a_i is able to function as a "decision maker", helping to classify test data. A hidden neuron with a large σ however is incapable of indicating the differences among test data, because its response is eventually random.

Substituting the continuous stochastic neuron into the restricted architecture of the PoE/RBM (Fig.3.1) forms the *Continuous Restricted Boltzmann Machine*. The CRBM is inherently a continuous-valued probabilistic model. Let the combination of all neurons' outputs $\{s_i\}$ represent the state of the CRBM. $\{s_i\}$ is determined according to the probability function in Eq.(4.4). The two-layer restricted architecture of the CRBM renders neurons in the same layer conditionally independent, given the outputs of neurons in the other layer. Neurons in the same layer can thus be updated simultaneously, and the dynamics becomes simply Gibbs sampling of visible and hidden layers alternately. It is necessary to prove that such dynamics lead the CRBM to equilibrium, before trying to derive any training rule to train the "equilibrium distribution" of the CRBM. The next subsection will thus prove the existence of equilibrium first. The similarity between the neuron of the CRBM (Fig.4.1) and the neuron of the Diffusion Network (Fig.2.9) implies that the CRBM is a form of Diffusion Network. The training rules for the CRBM may thus be derived from modifying the training rules of the Diffusion Network. After proving the existence of equilibrium, Sec.4.2 will discuss the relationship between the CRBM and the Diffusion Network, and Sec.4.3 will then derive the training rules for the CRBM.

4.1.2 The existence of equilibrium distribution

To prove that the equilibrium distribution of the CRBM exists, the dynamics of the CRBM can be considered as a *Markov chain* process, whose definition is as the following. Consider the stochastic process of a system over discrete time n ($n = 0, 1, 2, \ldots$). If the probability of the system's state at time n merely depends on the system's state at time $(n-1)$, the stochastic process of the system constitutes a *Markov chain*. Let S^n denote the state of the system at time n. Given that $S^{n-1}, S^{n-2}, \ldots, S^1, S^0$ have occurred, the Markov chain process is described more precisely by the following equation

$$Pr(S^n \mid S^{n-1}, S^{n-2}, \ldots, S^1, S^0) = Pr(S^n \mid S^{n-1}) \tag{4.5}$$

As Eq.(4.2) and Eq.(4.4) clearly indicate that, in a CRBM, the output probability of neurons at n-th step of Gibbs sampling merely depends on the outputs of other neurons at $(n-1)$-th step, the dynamics of the CRBM are proved to constitute a Markov chain process.

A Markov chain process is formulated in terms of a set of one-step transition probabilities, $\{P_{ij}(n, n-1)\}$. $P_{ij}(n, n-1)$ specifies the conditional probability that the system will be in state S_j at time n, given that the system is in state S_i at time $(n-1)$. The Markov chain is said to be *homogeneous* if the transition probabilities $\{P_{ij}(n, n-1)\}$ are independent of n. Let $P_{ij}(n, n-1)$ of a homogeneous Markov chain be abbreviated as P_{ij}. The Markov chain is said to be *ergodic* if $\{P_{ij}\}$ satisfies

$$P_{ij} > 0 \qquad \text{where} \quad i, j \in \textit{all possible states} \tag{4.6}$$

An *ergodic* Markov chain process can then be shown to converge to a unique solution, by the following theorem. The proof for the theorem is given in [66].

Theorem : Let $P_{ij}^{(n)}$ denote the probability that the state will be S_j after n steps of transition, given the initial state is S_i. For an *ergodic* Markov chain, there exists a solution for each

$$\Pi_j = \lim_{n \to \infty} P_{ij}^{(n)} \tag{4.7}$$

where $\{\Pi_j\}$ are unique and non-negative probabilities satisfying $\sum \Pi_j = 1$.

Note that only j remains in the subscript of Π, indicating the equilibrium state Π_j is independent of initial state S_i. Therefore, if the Markov chain corresponding to the dynamics of the CRBM is *ergodic*, the equilibrium of the CRBM exists.

As the dynamics of a CRBM are driven by Gibbs sampling of visible and hidden neurons alternately, the transition probability of a CRBM's state is proportional to the product of the output probabilities of individual neurons. Eq.(4.4) indicates that the output probability of an individual neuron is independent of time n, i.e. the number of steps of Gibbs sampling. The transition probability of the CRBM's state is thus also independent of time n. In other words, the Markov chain process of the CRBM's dynamics is *homogeneous*. Furthermore, both the Gaussian function and the derivative of the sigmoid function in Eq.(4.4) are nonnegative, indicating that the condition in Eq.(4.6) is satisfied. The Markov chain is thus *ergodic*, and equilibrium of the CRBM is guaranteed to exist.

4.2 The CRBM and the Diffusion Network

Both the CRBM and the Diffusion Network employ continuous stochastic neurons, and the neurons in both models rely on the input of Gaussian noise to achieve continuous stochastic

behaviour, as indicated by Fig.2.9 and Fig.4.1. The dynamics of the CRBM involve Gibbs sampling neurons according to Eq.(4.4), and are therefore a *discrete-time* stochastic process. Although Sec.2.4 described the dynamics of the Diffusion Network as a *continuous-time* stochastic process , the Diffusion Network actually has to be simulated in a *discrete-time* manner in software simulation. In the Diffusion Network, let $x_i^D(t)$ denote the pre-sigmoid input of neuron i (i.e. the total input before sigmoid function in Fig.2.9), and $\mu_i^D(t)$ the deterministic input component. Writing Eq.(2.15) as a discrete-time diffusion process for a finite time increment Δt gives [41]

$$x_i^D(t+\Delta t) = x_i^D(t) + \mu_i^D(t)\cdot\Delta t + \sigma\cdot z_i(t)\cdot\sqrt{\Delta t} \tag{4.8}$$

where $z_i(t)$ is a Gaussian random variable with zero mean and unit variance. As long as Δt is small enough [2], the discrete-time diffusion process above approximates the continuous-time diffusion process in Eq.(2.15). Eq.(4.2) indicates that the pre-sigmoid input x_i^C of the CRBM neuron is

$$x_i^C[n+1] = \sum_j w_{ij}s_j[n] + \sigma\cdot z_i[n] \tag{4.9}$$

where n indexes the step of Gibbs sampling. The equation above can be re-written in terms of discrete-time process as

$$\begin{aligned} x_i^C(t+\Delta t) &= x_i^C(t) + \mu_i^C(t)\cdot\Delta t + \sigma'\cdot z_i(t)\cdot\sqrt{\Delta t} \\ \text{with}\quad \mu_i^C(t) &= \kappa_i^C\cdot\left(\sum_j w_{ij}s_j(t) - x_i(t)\right) \end{aligned} \tag{4.10}$$

where κ_i^C is a constant and $\sigma' = \sigma/\sqrt{\Delta t}$. If $\Delta t = 1/\kappa_i^C$, the terms in $x_i^C(t)$ cancel such that Eq.(4.10) becomes the same as Eq.(4.9). Eq.(4.8) and Eq.(4.10) indicate that, in terms of discrete-time process, the CRBM is a form of the Diffusion Network with restricted, symmetric connection.

[2] [20] defines the *bandwidth* (BW) of a stochastic differential equation. A small Δt refers to $\Delta t \approx \frac{1}{100}\left(\frac{1}{BW}\right)$

Though the equilibrium of the CRBM is proved to exist, it is generally difficult to define the explicit form of the equilibrium of a Markov chain process. This is similar in the case of the diffusion process for the Diffusion Network [27], as described in Sec.2.4.1. Nevertheless, Movellan specified the condition for a symmetric Diffusion Network to have an explicit form of equilibrium, the *continuous Boltzmann distribution*, and subsequently derived the algorithm for training the continuous Boltzmann equilibrium [27], as described in Sec.2.4. It is thus necessary to investigate the conditions under which the CRBM approximates a symmetric Diffusion Network with continuous Boltzmann equilibrium.

Consider the symmetric Diffusion Network with the neuron in Fig.2.9, and let $s_i = \varphi_i(x_i)$. The energy function in Eq.(2.23) for the symmetric Diffusion Network is re-written as

$$E_{DN} = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \frac{\rho_i}{a_i} \int_{s_0}^{s_i} \phi^{-1}(s) ds \tag{4.11}$$

In order to have continuous Boltzmann equilibrium, the drift component of neurons should equal the negative gradient of the energy, i.e. the drift component of neuron i should be

$$\begin{aligned} \mu_i^{DN} &= -\frac{\partial E_{DN}}{\partial x_i} = -\frac{\partial E_{DN}}{\partial s_i} \cdot \frac{\partial s_i}{\partial x_i} \\ &= a_i \cdot \phi'(x_i) \cdot \left(\sum_j w_{ij} s_j - \frac{x_i}{a_i} \right) \end{aligned} \tag{4.12}$$

Substituting μ_i^{DN} into the discrete-time diffusion process in Eq.(4.8), and incorporating the capacitance parameter C_i of the neuron, μ_i^D in Eq.(4.8) for the symmetric Diffusion Network is

$$\begin{aligned} \mu_i^D &= \kappa_i^D \cdot \mu_i^{DN} \\ &= \kappa_i^D \cdot a_i \cdot \phi'(x_i) \cdot \left(\sum_j w_{ij} s_j - \frac{\rho_i x_i}{a_i} \right) \end{aligned} \tag{4.13}$$

where $\kappa_i^D = 1/C_i$. Comparing μ_i^D with μ_i^C in Eq.(4.10) gives the conditions under which the CRBM approximates the symmetric Diffusion Network with continuous Boltzmann equilibrium.

Three sets of conditions, from "strict" to "soft", for the approximation to hold can be defined as follows. The strictest condition is

Condition A

$$\phi'(x_i) = \text{constant} \quad \wedge \quad a_i = 1 \tag{4.14}$$

The symbol $\wedge$ denotes the Boolean operation "AND" between two sub-conditions. Under condition A, let $\Delta t_D = 1/(\kappa_i^D \phi')$, $\Delta t_C = 1/\kappa_i^C$, and $\rho_i = 1$. According to Eq.(4.10) and Eq.(4.13), it is obvious that $\mu_i^D \cdot \Delta t_D = \mu_i^C \cdot \Delta t_C$, i.e. the drift components of both the CRBM and the symmetric Diffusion Network are exactly the same. Since the two models have the same terms of Gaussian-noise disturbances, the CRBM under condition A is equivalent to the symmetric Diffusion Network with continuous Boltzmann equilibrium. Figure 4.3 shows the sigmoid function $\phi(x_i)$ with asymptotes at ± 1, together with the derivative of the sigmoid function $\phi'(x_i)$. $\phi'(x_i)$ approximates a constant value when x_i is small, and equals 0 as x_i goes to infinity. The latter case is undesirable because $\phi'(x_i) = 0$ makes $\Delta t_D = 1/(\kappa_i^D \phi')$ become infinity. Fig.4.3 reveals that $\phi'(x_i)$ approximates a constant value within $[0.4, 0.5]$ when $-1 < x_i < 1$. Therefore, condition A is satisfied only when $\{x_i\}$ of all neurons are relatively small and $a_i = 1$ for all neurons.

$1/\rho_i$ represents the resistance of neuron i, and is thus allowed to vary from one to another neuron. Eq.(4.13) suggests that setting $\rho_i = a_i$ rather than $\rho_i = 1$ yields a softer condition as follows

Condition B

$$\phi'(x_i) = \text{constant} \quad \wedge \quad \rho_i = a_i \quad \wedge \quad a_i \ll \kappa_i \tag{4.15}$$

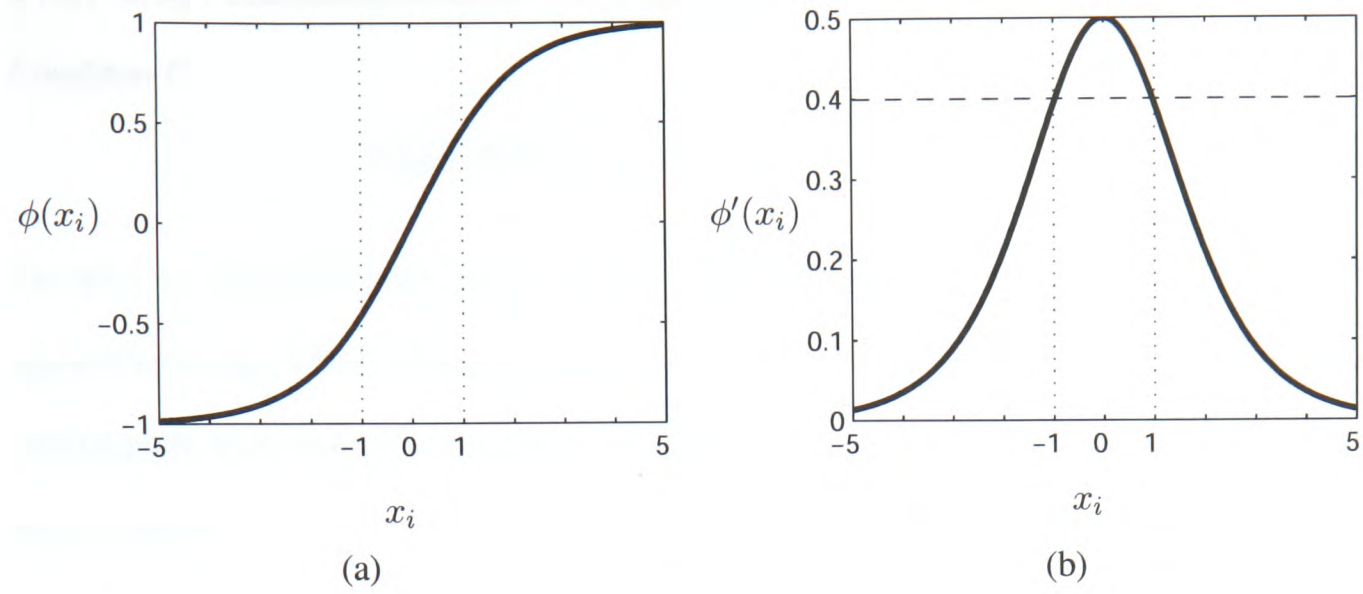


Figure 4.3: *(a)The sigmoid function $\phi(x_i)$ with $\theta_H = -\theta_L = 1$. (b)The derivative of the sigmoid function in (a), $\phi'(x_i)$*

Let $\Delta t_D = 1/(a_i \kappa_i^D \phi')$ and $\Delta t_C = 1/\kappa_i^C$. The third sub-condition $a_i \ll \kappa_i^D$ renders the variation of Δt_D negligible. This sub-condition is normally satisfied, because it generally requires $\kappa_i^D \gg 0$ to ensure that Δt_D is small. With the Δt_D and Δt_C specified above, Eq.(4.10) and Eq.(4.13) indicate that $(\mu_i^D \cdot \Delta t_D)$ "approximately" equals $(\mu_i^C \cdot \Delta t_C)$ under condition B. In other words, when condition B is satisfied, the CRBM "approximates" the symmetric Diffusion Network with continuous Boltzmann equilibrium. The word "approximate" remarks the fact that Δt_D merely approximates a constant value. The main advantage of condition B over condition A is that parameter a_i can vary from one to another neuron, enriching the stochastic behaviour of neurons and thus the modelling flexibility, as discussed in Sec.4.1.1. Ch.5 and Ch.6 will show that the range of a_i is limited to $[0.5, 4]$ in hardware implementation, but is enough to provide essential modelling flexibility. Condition B is therefore satisfied when the $\{x_i\}$ of all neurons are relatively small and the range of a_i is limited to ensure an approximately-constant Δt_D. The latter normally holds for the hardware implementation of the CRBM.

It is possible to further "soften" the strict sub-condition "$\phi'(x_i) =$ constant". Assume $\phi'(x_i)$ has a limited range. $\Delta t_D = 1/(a_i \kappa_i^D \phi')$ still approximates a constant value, provided

$\phi'(x_i) \ll \kappa_i^D$. The softest condition is thus given as

Condition C

$$\phi'(x_i) \ll \kappa_i^D \quad \wedge \quad \rho_i = a_i \quad \wedge \quad a_i \ll \kappa_i^D \tag{4.16}$$

Let $\Delta t_D = 1/(a_i \kappa_i^D \phi')$ and $\Delta t_C = 1/\kappa_i^C$. When condition C is satisfied, the CRBM also approximates the Diffusion Network with continuous Boltzmann equilibrium, although the approximation error under this condition is greater than under condition B. Let $s = \phi(x_i)$. It is easy to derive

$$\phi'(x_i) = \frac{(1 - s^2)}{2} \tag{4.17}$$

Fig.4.3(b) shows that $\phi'(x_i)$ equals the maximum value 0.5 when $s = 0$, while decreasing to 0.095 when $s = \pm 0.9$. The ratio between the maximum and minimum of $\phi'(x_i)$ therefore approximately equals 5 when $-0.9 < s < 0.9$. This implies that, as long as a neuron's output range lies in $[-0.9, 0.9]$, the first sub-condition $\phi'(x_i) \ll \kappa_i^D$ in Eq.(4.16) is satisfied. Condition C thus not only allows a_i to have various values, but also allows a reasonable value range of $\{x_i\}$ for all neurons.

Condition A implies that the equilibrium of the CRBM is never equal to continuous Boltzmann equilibrium, while condition C indicates that the equilibrium of the CRBM still "approximates" continuous Boltzmann equilibrium if both $\{s_i\}$ and $\{a_i\}$ of all neurons have limited range. Condition C is violated seriously only when the outputs of neurons approach the asymptotes of the sigmoid function (e.g. $s_i = \pm 1$), resulting in $\phi' = 0$ and thus an implausible Δt_D. For us, the primary purpose of comparing the CRBM with the symmetric Diffusion Network is to derive the training rules for the CRBM, not to prove that the equilibrium of the CRBM is identical to the continuous Boltzmann equilibrium. Therefore, the main concern is whether the approximation holds during a training process, such that it is plausible to derive the CRBM's training rules by modifying Eq.(2.21), the training rules of the symmetric Diffusion Network.

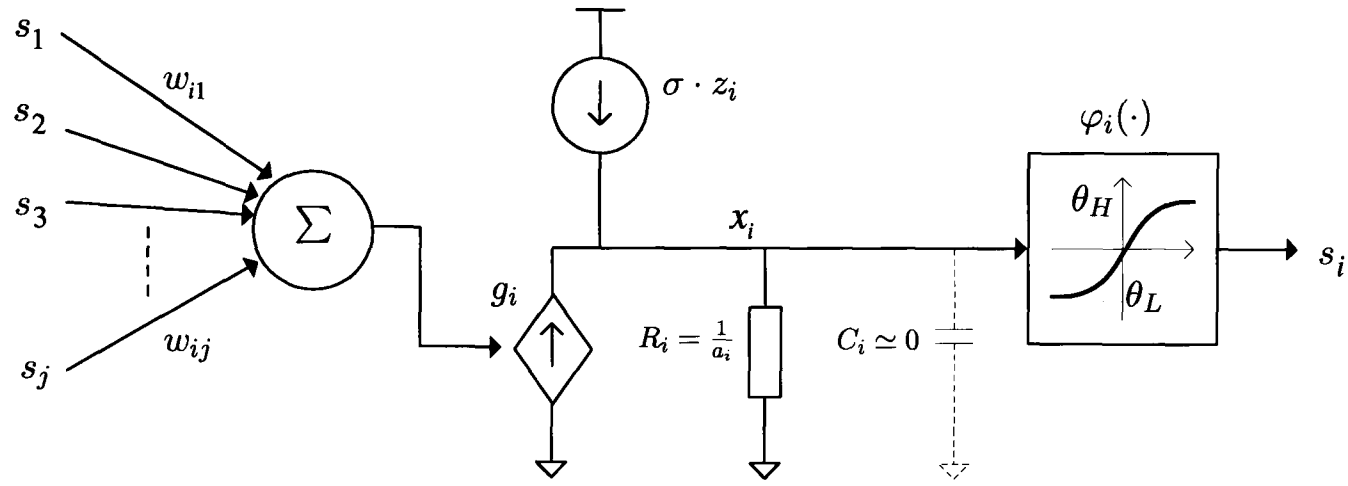


Figure 4.4: *The particular neuron with which a symmetric Diffusion Network is approximated by the CRBM under condition C.*

During a training process, the output range of visible neurons can be easily confined to a smaller range by normalising training data into $[-0.9, 0.9]$ rather than $[-1, 1]$, for example. As the CRBM will be trained by the minimising-contrastive-divergence (MCD) method, it can be assumed that the output range of hidden neurons seldom reaches the asymptotes during one-step Gibbs sampling. Under this assumption, condition C is satisfied during most periods of a training process. Taking another perspective, condition C constrains a neuron to have near-zero capacitance ($C_i \approx 0$) and resistance $R_i = 1/a_i$, indicating that the CRBM actually approximates the symmetric Diffusion Network with the particular neuron shown in Fig.4.4. The neuron in Fig.4.4 merely differs from the CRBM neuron in Fig.4.1 by the additional resistor R_i. In other words, R_i introduces the approximation error for condition C to be violated. As MCD method requires only one step of Gibbs sampling, the error introduced by R_i can be assumed to be negligible in general. According to Sec.2.6, the MCD method is applicable to the training rule of the symmetric Diffusion Network. It is thus reasonable to derive a training rule for the CRBM from that of the symmetric Diffusion Network.

4.3 Hardware-amenable training algorithm

Sec.2.4.3 gives the training rule for a symmetric Diffusion Network as

$$\triangle\lambda_i = \langle S_{\lambda_i}\rangle_0 - \langle S_{\lambda_i}\rangle_\infty \tag{4.18}$$

where λ_i represents any parameter of the model, and S_{λ_i} is the system covariate. Eq.(2.24) and Eq.(2.25) give the system covariate for w_{ij} and a_i, respectively. Drawing on the Minimising-Contrastive-Divergence (MCD) method, Eq.(4.18) can be further simplified as the following equation [68], avoiding a time-consuming relaxation search for equilibrium.

$$\triangle\hat{\lambda}_i = \langle S_{\lambda_i}\rangle_0 - \langle S_{\lambda_i}\rangle_1 \tag{4.19}$$

where $\langle\cdot\rangle_1$ denotes the expectation value over one-step Gibbs-sampled data. Substituting the system covariate in Eq.(2.24) and Eq.(2.25) into Eq.(4.19) then gives the MCD training rules for both the weights $\{w_{ij}\}$ and the noise-control parameters $\{a_i\}$ of the CRBM [68].

$$\Delta\hat{w_{ij}} = \eta_w(\langle s_i \cdot s_j\rangle_0 - \langle s_i \cdot s_j\rangle_1) \tag{4.20}$$

$$\Delta\hat{a_i} = \eta_a\left(\left\langle\frac{1}{a_i^2}\int_{s_0}^{s_i}\phi^{-1}(s)ds\right\rangle_0 - \left\langle\frac{1}{a_i^2}\int_{s_0}^{s_i}\phi^{-1}(s)ds\right\rangle_1\right) \tag{4.21}$$

η_w and η_a are the learning rates of $\{w_{ij}\}$ and $\{a_i\}$, respectively, and $s_0 = \phi(0)$. Let s_i^0 denote initial data, and s_i^1 one-step Gibbs-sampled data. Eq.(4.21) is re-written as

$$\Delta\hat{a_i} = \eta_a\left(\frac{1}{a_i^2}\left\langle\int_{s_i^1}^{s_i^0}\phi^{-1}(s)ds\right\rangle\right) \tag{4.22}$$

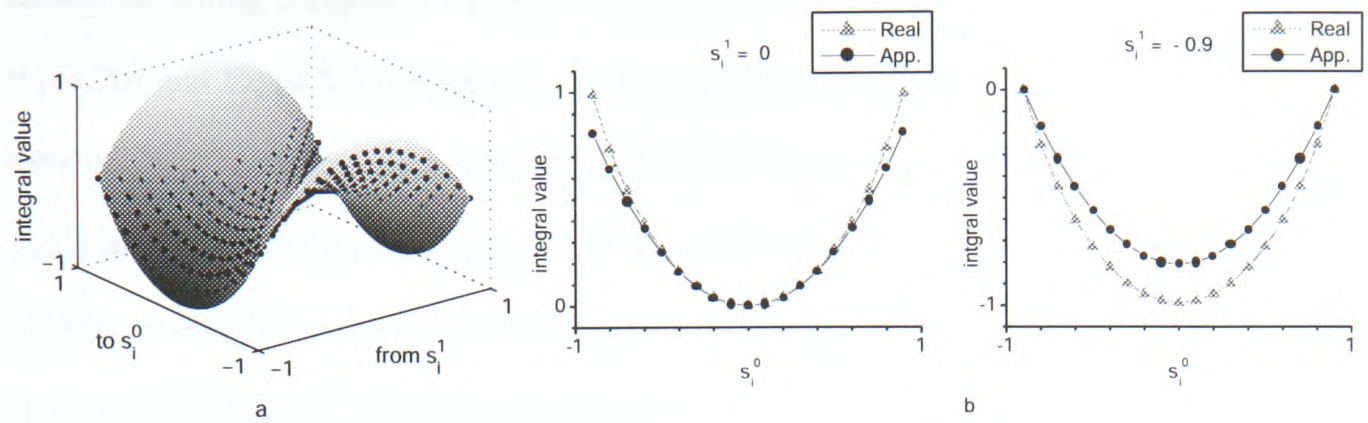


Figure 4.5: *(a)Approximation of Eq.(4.23) over the value ranges of s_i^0 and s_i^1. (b) Two slices at $s_i^1 = 0$ (left) and at $s_i^1 = -0.9$ (right).*

$\langle\cdot\rangle$ denotes the expectation value over training data. To simplify the training rule for $\{a_i\}$, primarily for ease of hardware design, the integral term in Eq.(4.22) is further approximated as

$$\int_{s_i^1}^{s_i^0} \phi^{-1}(s)ds \propto (s_i^0 + s_i^1)(s_i^0 - s_i^1) \tag{4.23}$$

The accuracy of this approximation is illustrated by Fig.4.5. The horizontal axes in Fig.4.5(a) span the full ranges of s_i^0 and s_i^1. The continuous surface shows the values of the real integral and the black points illustrate the approximate values given by Eq.(4.23), respectively. Owing to the small difference between the real and approximate values, the black points are invisible over some ranges. Fig.4.5(b) shows two slices of the 3D plot, for viewing the difference clearly. Fig.4.5 reveals that the difference is small and encourages the use of the approximation. From Eq.(4.23) and Eq.(4.22), the simplified training rule for $\{a_i\}$ is

$$\Delta\hat{a}_i = \frac{\eta_a}{a_i^2}\left(\langle s_i^2\rangle_0 - \langle s_i^2\rangle_1\right) \tag{4.24}$$

Eq.(4.20) and Eq.(4.24) indicates that the training rules for the CRBM requires simple subtraction and multiplication operations. Updating each parameter w_{ij} requires merely the outputs of the two neurons connected by w_{ij}. Such a simple and localised training rule reduces signif-

icantly the wiring complexity in hardware implementation. In addition, the similarity between Eq.(4.20) and Eq.(4.24) is striking. As the MCD training rule for $\{w_{ij}\}$ has been successfully implemented and tested in VLSI [51, 52], the similarity implies that the training rules for $\{w_{ij}\}$ and $\{a_i\}$ may be implemented with the same circuit. Therefore, the training rules for the CRBM are suitable for hardware implementation. Before engaging in the hardware implementation of the CRBM, the CRBM's performance on modelling continuous data with the training rules above is examined with both artificial and real heart-beat data.

4.4 Experiments with artificial data

4.4.1 Training data with a non-symmetric distribution

For comparison with the PoE/RBM and the RBMrate, a CRBM with two visible neurons and four hidden neurons was trained to model the training data in Fig.4.6(a). The CRBM was trained with $\eta_w = 1.5$, $\eta_a = 1$, $\theta_H = -\theta_L = 1$, and $\sigma = 0.2$ for all neurons. After 4000 training epochs, the CRBM reconstructed data points as shown in Fig.4.6(b). The reconstruction was obtained by Gibbs sampling from 400 random initial data for 20 steps. Fig.4.6(b) shows that the CRBM models the data well and reconstructs continuous-valued data points with no artificial quantisation effect. The statistical density over the visible-state space was estimated through the following steps:

1. Collect 1000 samples from the trained CRBM's generative model.
2. Round the (x,y) coordinates of these sampled data into the same 0.04 unit grid as the RBMrate used in Sec.3.2.2.
3. Count the number of data at each grid point.
4. Use the occurrences to estimate probabilities over the visible-state space.

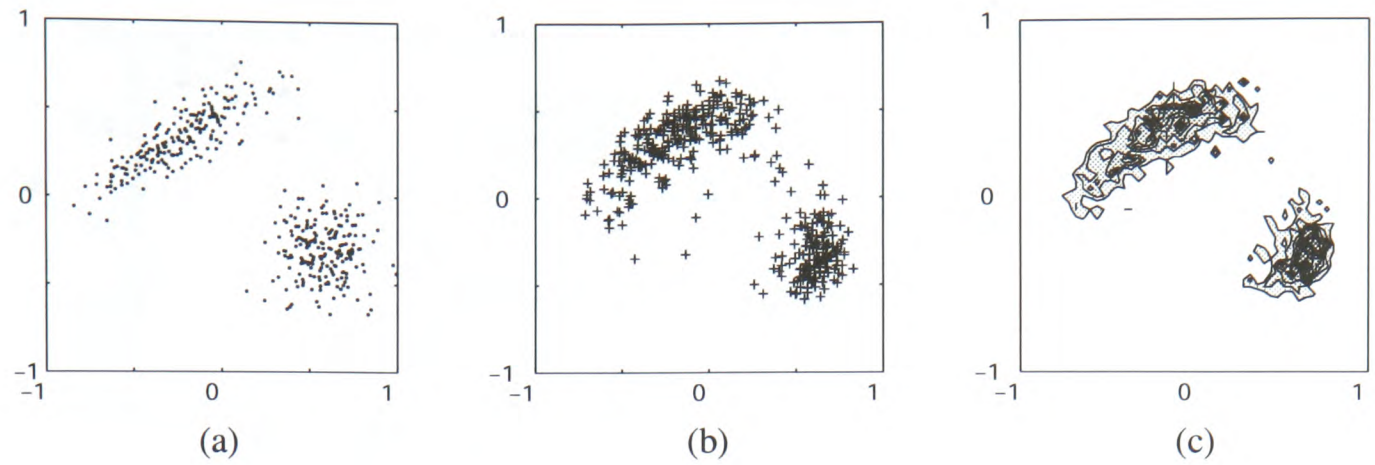


Figure 4.6: *(a)The training data containing two clusters of 200 data points in Gaussian distribution. One Gaussian is elliptical, and the other circular (b)The 20-step reconstruction generated by the trained CRBM from 400 random initial input. (c)The estimated statistical density over the visible-state space, assigned by the trained CRBM. The darker colour corresponds to a higher probability*

As these discrete-valued data correspond exactly to all possible states of the RBMrate used in Sec.3.2.2, taking the rounded values not only eases the density estimation over continuous-valued space but also facilitates the comparison between the two models. Fig.4.6(c) shows the estimated statistical density over the visible-state space. Obviously, the CRBM defines a smoother density distribution than does RBMrate. One elliptic Gaussian and one circular Gaussian are shown, indicating that the CRBM captures the distribution of the training data.

4.4.2 Evolution of the noise-control parameter

Unlike the PoE/RBM and the RBMrate, the CRBM has not only adaptive weights $\{w_{ij}\}$ but also adaptive noise-control parameters $\{a_i\}$ during training. Both $\{w_{ij}\}$ and $\{a_i\}$ were adapted in parallel, according to Eq.(4.20) and Eq.(4.24). While the weights $\{w_{ij}\}$ learnt to encode the data distribution, it is interesting to observe what the noise-control parameters $\{a_i\}$ learn. Fig.4.7 (a) and (b) show the evolution of the $\{a_i\}$ for visible and hidden neurons during training. To perceive how a neuron's stochastic behaviour changes with parameter a_i, Fig.4.8(a) displays sigmoid functions with various values of a_i, together with the probability density of Gaussian

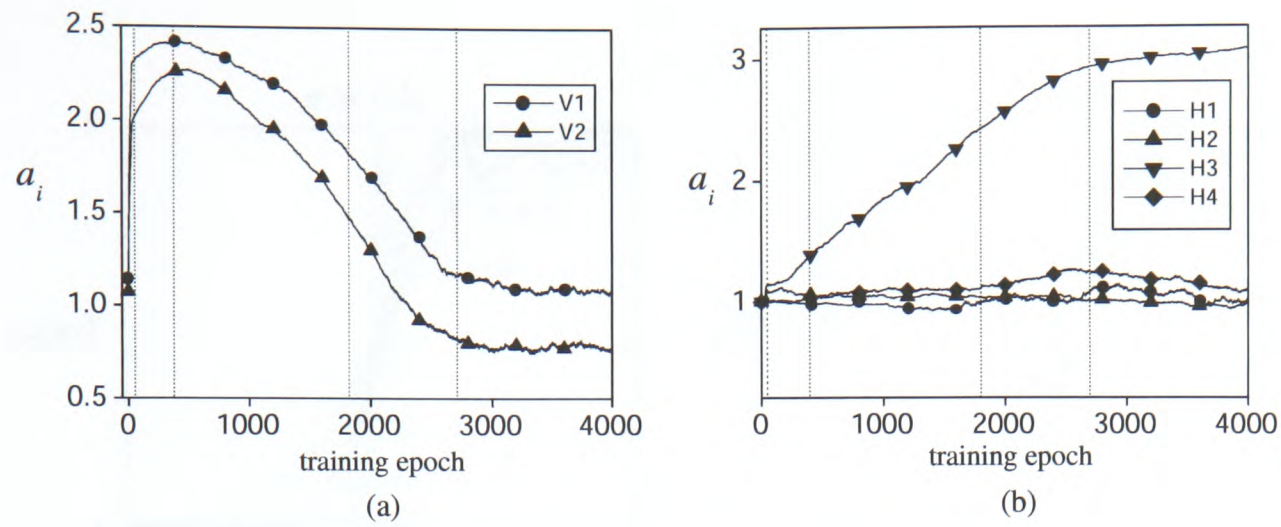


Figure 4.7: *Evolution of the noise-control parameters for (a) visible neurons and (b) hidden neurons, during the training process in Sec.4.4.1.*

noise when $\sigma = 0.2$ (the grey curve in Fig.4.8(a)).

The traces in Fig.4.7 show clearly how the data model of Fig.4.6 forms during training. As all $\{a_i\}$ were initially set to 1, the $\{a_i\}$ of the visible neurons displayed a form of "autonomous annealing" driven by Eq.(4.24), gradually reducing the level of noise and thus the neurons' tendency to be binary. The gradual reduction of $\{a_i\}$ is called "annealing" because its effect on the CRBM is similar to that of the temperature factor on the Boltzmann Machine, as described in Sec.2.3.1. The a_i for the hidden neuron H3 rose from $a_{H3} = 1$ to $a_{H3} = 3$, while a_{H1}, a_{H2}, and a_{H4} remained ≈ 1. So the hidden neurons clearly perform different functions in the model.

To investigate what this behaviour means in terms of model formation during training, 20-step reconstructions by the CRBM after 50, 395, 1800, and 2700 training epochs are shown in Fig.4.9. The dotted lines in Fig.4.7 highlight the corresponding values of $\{a_i\}$. Very early in the training, the $\{a_i\}$ of both visible neurons rises abruptly to form *near-binary* neurons, allowing a wide "search" of the solution space to find crude binary-approximations to the training data clusters, as illustrated by Fig.4.9(a). This reconstruction is similar to the reconstruction

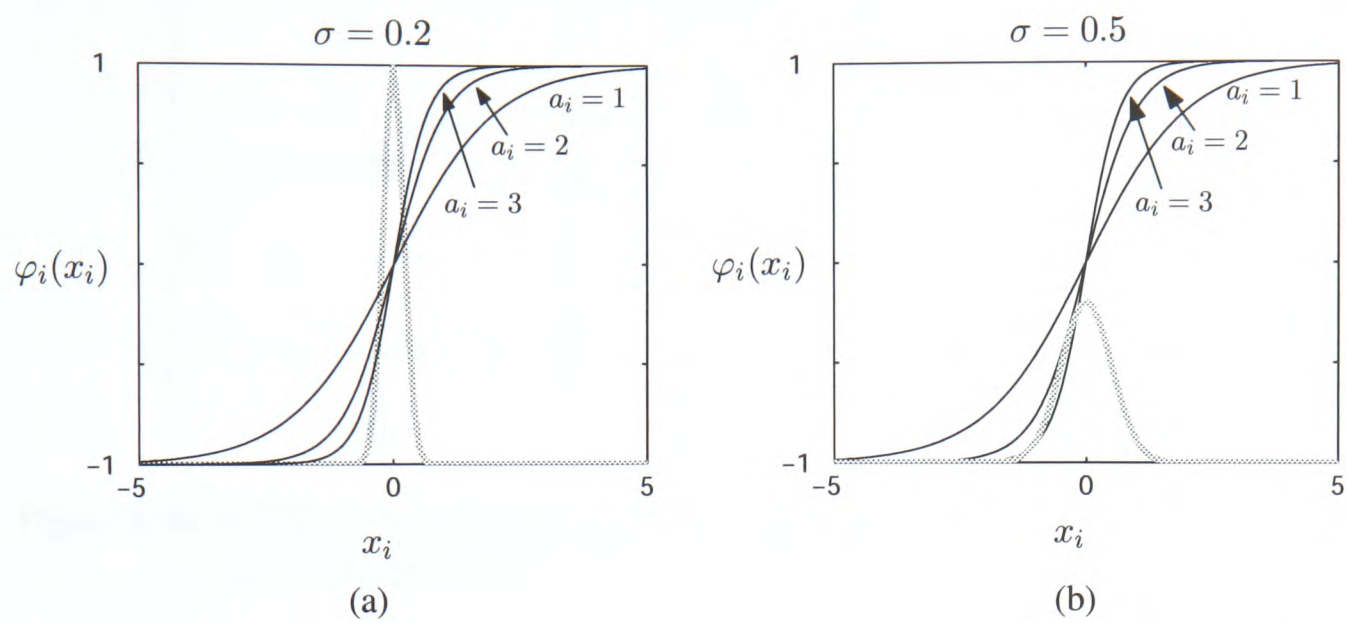


Figure 4.8: *The sigmoid functions with various values of a_i are plotted with the probability density function of Gaussian noise (grey curves) with (a) $\sigma = 0.2$ (b) $\sigma = 0.5$*

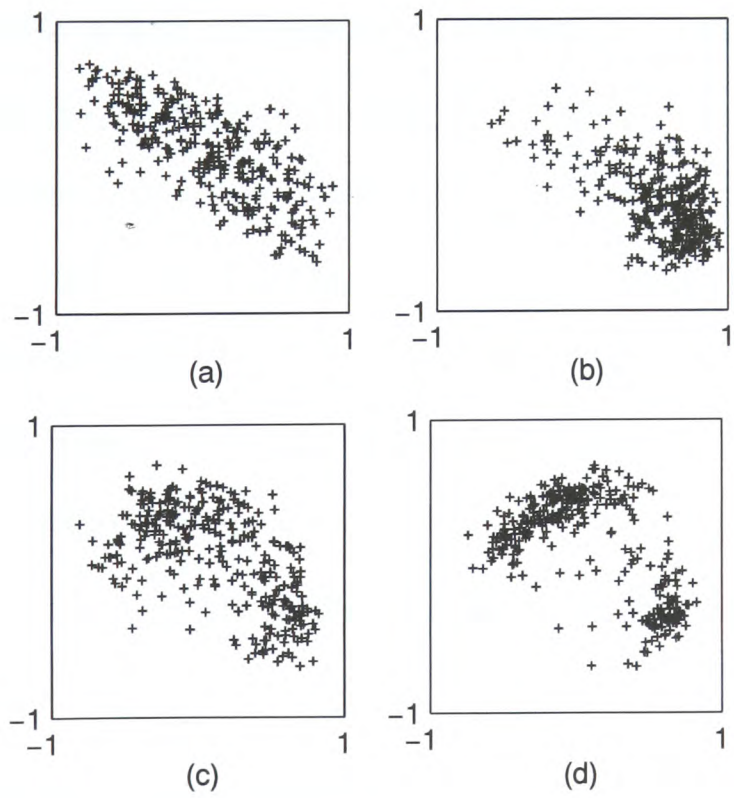


Figure 4.9: *20-step Gibbs sampled reconstructions by the CRBM (a) after 50 training epochs (b) after 395 epochs (c) after 1800 epochs (d) after 2700 epochs*

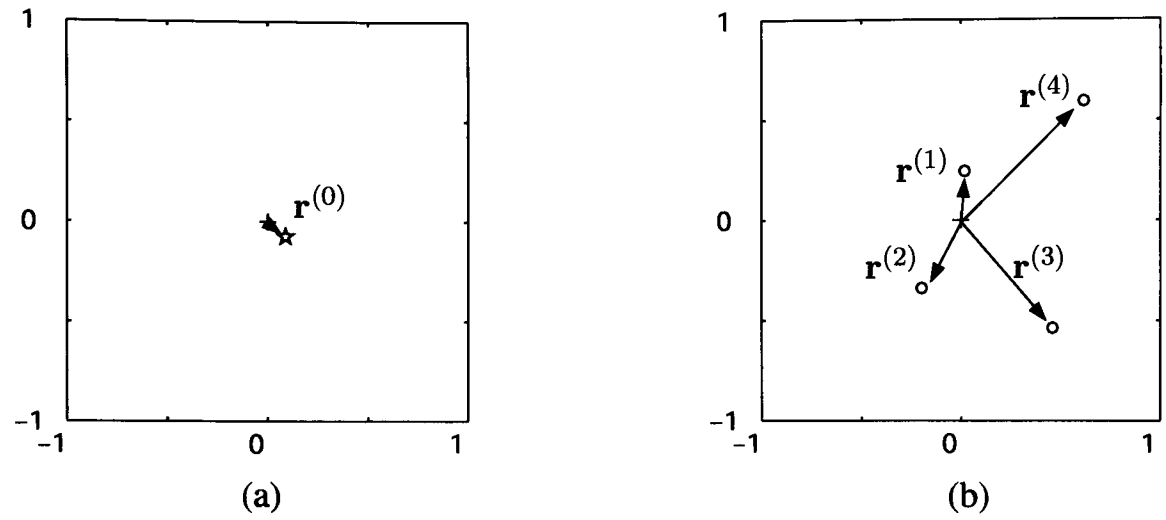


Figure 4.10: *The learnt weight vectors of (a) the hidden bias unit (b) the hidden neurons, after* 4000 *training epochs.*

in Fig.3.9, generated by the PoE/RBM with binary-stochastic neurons. The $\{a_i\}$ of visible neurons peaks at epoch 395. Fig.4.9(b) reveals that the large value of $\{a_i\}$ compresses the reconstruction and inhibits the adaptation of the weights $\{w_{ij}\}$. The $\{a_i\}$ of visible neurons subsequently decreases, allowing more flexible adaptation of the $\{w_{ij}\}$. As training progresses further, the $\{a_i\}$ of visible neurons continues to fall, enabling the CRBM to model continuous data with more deterministic visible neurons. During this adaptation process, a_{H3} in the hidden layer rises, causing its neuron to become near-binary and allowing two clusters to be modelled clearly, as illustrated by the reconstructions on Fig.4.9(c) and 4.9(d). By epoch 2700, the values of $\{a_i\}$ for both visible and hidden neurons have essentially settled and the CRBM models the data distribution well, as shown in Fig.4.9(d). The weight parameters $\{w_{ij}\}$, though not shown, also settle after epoch 2700, indicating that the training reaches equilibrium. The results above indicate that Eq.(4.24) leads to intriguing and encouraging training behaviour in this stylised, but non-trivial example.

Fig.4.10 shows the learnt weight vectors of the hidden bias unit and the hidden neurons after 4000 training epochs. The near-binary hidden neuron H3 encodes the separation vector between the two clusters, while the other hidden neurons with smaller $\{a_i\}$ encode the variance

of the clusters in different directions. The hidden neuron H3 thus functions as a "decision maker", deciding which cluster each reconstructed point should belong to. At a particular step of reconstruction, if the output of H3 approximates $+1$, the reconstructed point will be in the circular cluster, while if the output of H3 approximates -1, the reconstructed point will be in the elliptic cluster. From another point of view, the hidden neuron H3 responds with significantly different outputs when data from different clusters are presented to the visible neurons. If a set of test data contains data points from both clusters, the test data can thus be easily classified according to the responses of the hidden neuron H3. This feature is inherited from the Restricted Boltzmann Machine, as described in Sec.2.5. Experiments with real heartbeat data will show that this feature is useful in detecting abnormal heartbeats.

4.4.3 Modelling the correlations of training data

In addition to compare the CRBM with other models, the CRBM's modelling capability is further explored by using the training data with a doughnut-shaped distribution, as shown in Fig.4.11(a). The inner circle of the doughnut shape has a radius of 0.6, and the outer circle has a radius of 0.8. To model the training data well, the CRBM has to capture the detailed correlations between the two dimensions of the training data. The term "detailed correlations" refer to the fact that the value range of one dimension closely relates to the value of the other dimension. For example, if $v_1 \approx 0$, the value of v_2 should be confined in $\Delta_1 = [-0.8, -0.6]$ or in $\Delta_2 = [0.6, 0.8]$, as illustrated by Fig.4.11(b), such that the reconstructed points (v_1, v_2) always locate in either the top or the bottom section of the circular band. If $v_1 \approx -0.7$, on the other hand, the values of v_2 must lie in $\Delta_3 = [-0.38, 0.38]$, in order to have reconstructed points on the left section of the circular band, as depicted in Fig.4.11(b). The relationship between v_1 and v_2 changes continuously and noticeably across the visible-state space, and hence demands more sophisticated modelling capability.

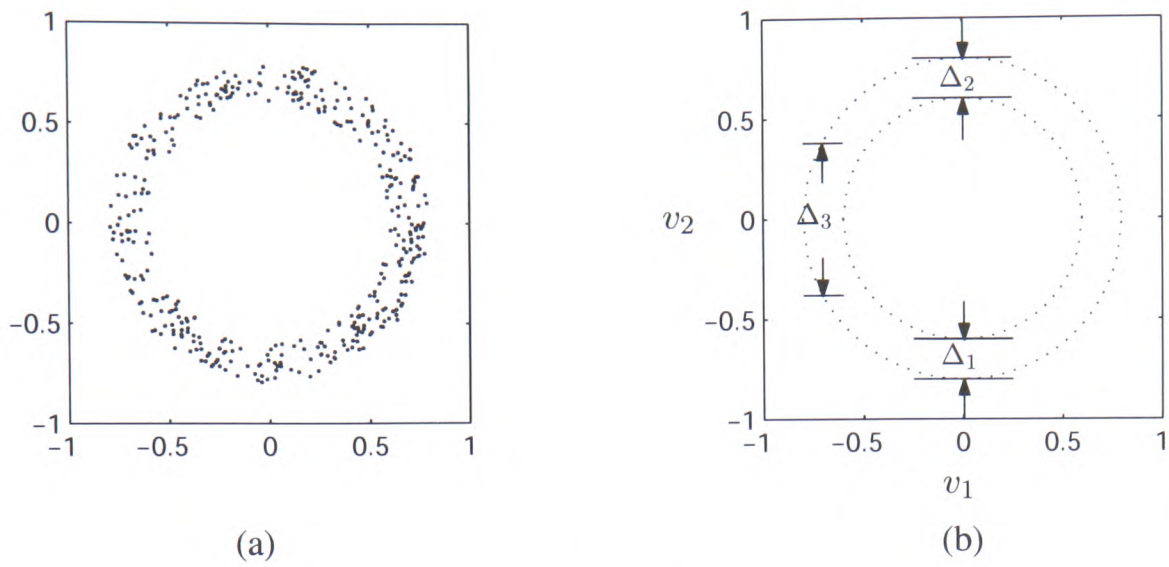


Figure 4.11: *(a)400 training data points with a doughnut-shaped distribution (b)The value range of v_2 strongly relates to the value of v_1.*

A CRBM with two visible and ten hidden neurons was trained with $\mu_w = 0.45$, $\mu_a = 1.5$, $\sigma = 0.05$ for visible neurons and $\sigma = 0.2$ for hidden neurons. After 20000 training epochs, the 20-step reconstruction by the trained CRBM is shown in Fig.4.12(a). The reconstructed data have almost the same distribution as the training data, indicating that the CRBM models the detailed correlations of the training data well. This promising result demonstrates the CRBM's rich representational power, attributable to the use of continuous stochastic neurons. In addition, this result further implies that the CRBM is potentially capable of solving nonlinear problems. Consider the nonlinear problem shown in Fig.4.12(b). The data points in the region with vertical lines in Fig.4.12(b) belong to category A, while the data points in the region with horizontal lines belong to category B. The two categories are only separable by drawing a circular line in the white circular band. This problem therefore differs from the task of distinguishing between the two elliptic clusters in Fig.4.6(a), as the two clusters in Fig.4.6(a) are linearly-separable. The experiment result in Fig.4.12(a) implies that the CRBM is capable of "drawing the circle", i.e. solving this particular nonlinear problem, although more simulations are necessary to examine the CRBM's ability to solve general nonlinear problems.

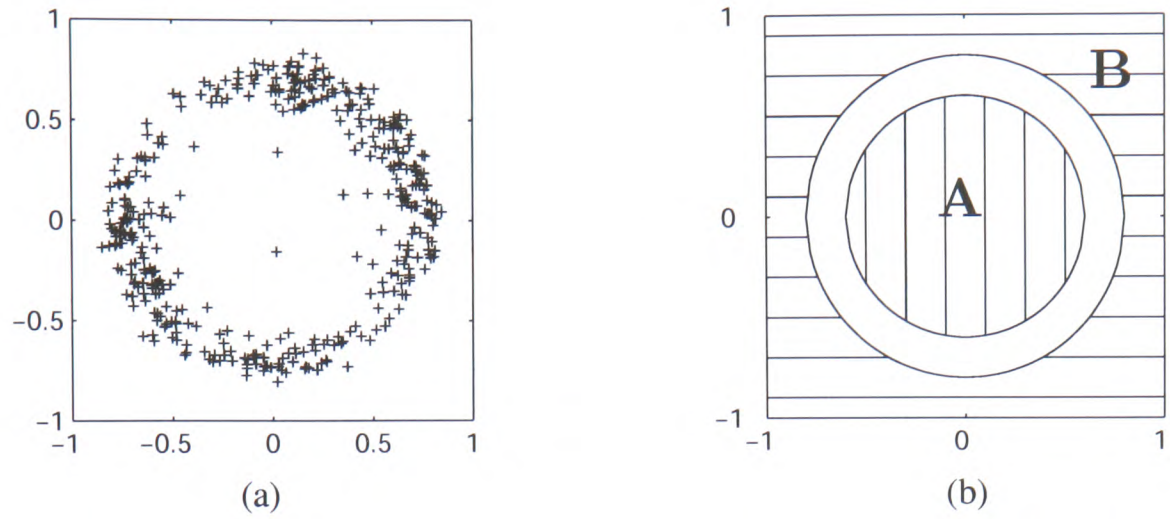


Figure 4.12: *(a)The reconstruction by the trained CRBM after 20 steps of Gibbs sampling from 400 random initial data. (b)A nonlinear problem of distinguishing between category A and B in the two-dimensional space.*

4.5 Experiments with real heartbeat data

To highlight the improved modelling richness of the CRBM and to give the results in Sec.4.4 credence, the CRBM was trained to model the heart-beat (ECG) data used in [25] and [69]. Training and test datasets, of 500 and 1700 heartbeats respectively, were extracted from a 30-minute ECG trace. Each heartbeat consists of 65 sampled values with amplitudes normalised into the range of $[-1, 1]$, and the ECG datasets contain both normal heartbeats (e.g. Fig.4.13(a)) and Ventricular Ectopic Beats (VEB) (e.g. Fig.4.13(b)). The 500 training data contain 6 VEBs, and the 1700 test data contain 27 VEBs.

A CRBM with 65 visible and four hidden neurons was trained for 4000 epochs with $\eta_w = 1.5$, $\eta_a = 1$, $\theta_H = -\theta_L = 1$, $\sigma = 0.2$ for visible neurons and $\sigma = 0.5$ for hidden neurons. Figure 4.13 shows the 20-step reconstruction generated by the trained CRBM with the input of (a) a normal QRS complex [3], typical of $\approx$ 99% of the training data and (b) a typical VEB. The CRBM modelled both forms of heartbeat successfully, although VEBs represent only 1% of the training data. Following [25], Fig.4.13(c) shows the receptive fields (i.e. the 65-dimensional

[3]The letters QRS refer to the different points in a heartbeat cycle when the heart is actually pumping [69].

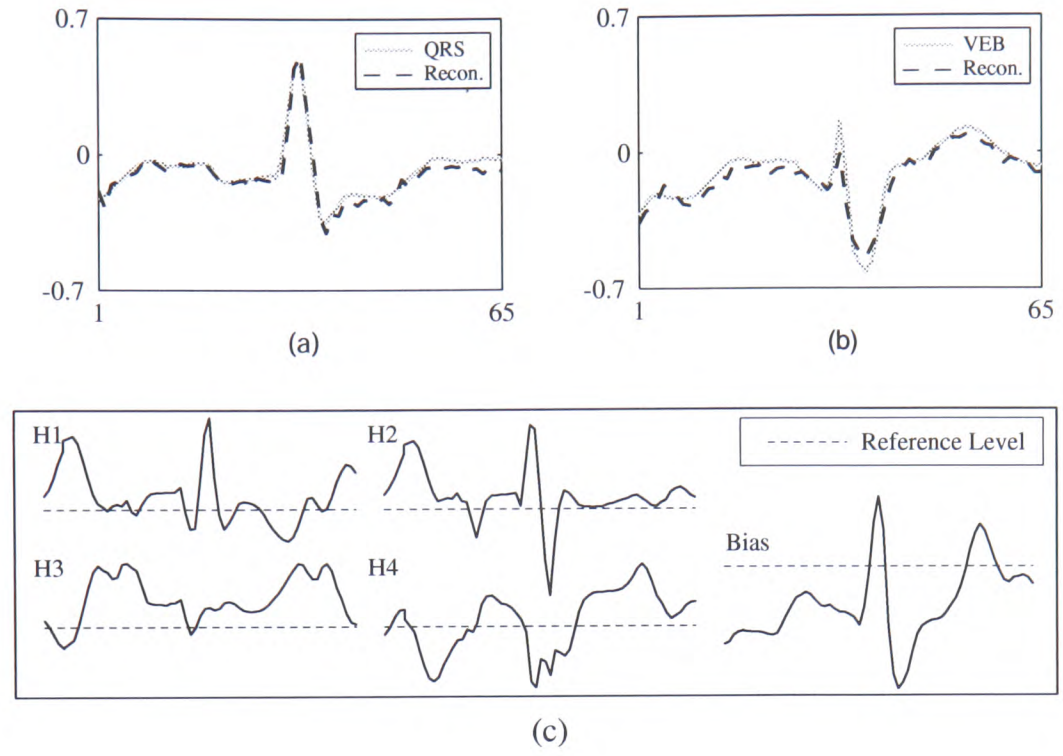


Figure 4.13: *Reconstruction by the trained CRBM and the receptive fields of the hidden bias unit and the four hidden neurons (a)Reconstruction from input of normal QRS complex (b)Reconstruction from input of typical VEB (c)Receptive fields of the hidden bias unit and the four hidden neurons* $H1 - H4$

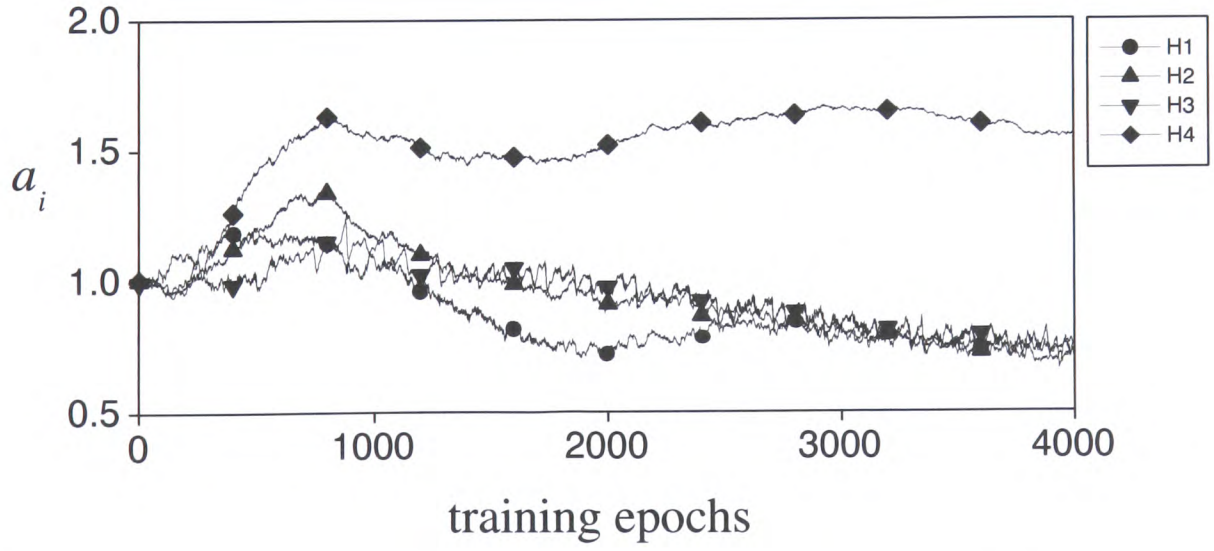


Figure 4.14: *The evolution of parameter* $\{a_i\}$ *for four hidden neurons during the training process of modelling ECG data.*

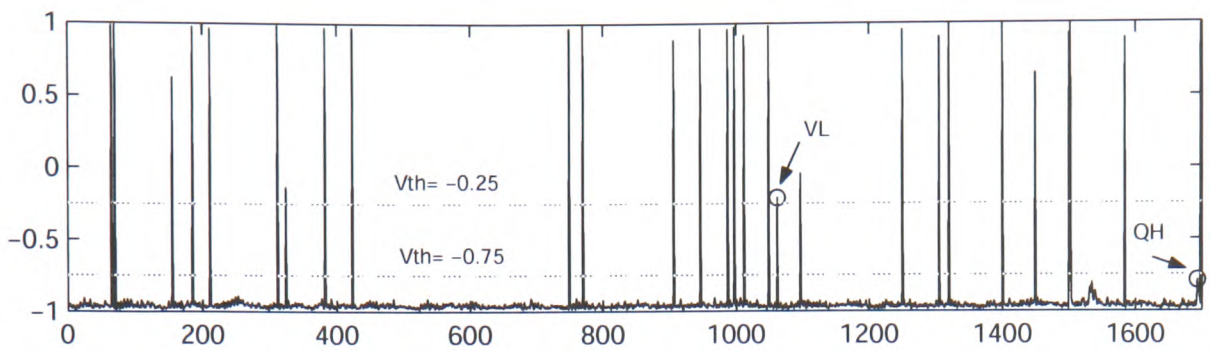


Figure 4.15: *Post-sigmoid activities of H4 corresponding to 1700 testing data*

weight vectors) of the hidden bias unit and the four hidden neurons. The bias unit codes an "average" normal QRS complex, and H3 adds amplitude and detailed structure to the P- and T-waves. H1 and H2 drive a small horizontal shift and a magnitude variation of the QRS complex. Finally, H4 encodes the significant dip found in a VEB. A CRBM with only two hidden neurons yielded similar success, although the ability to model inter-QRS variations was reduced. This modelling richness compares favourably with that of a PoE/RBM examined in [25], which merely modelled normal QRS complex.

The receptive field of H4 in Fig.4.13 suggests a simple and pragmatic approach to detecting the existence of VEB in a set of test data. When H4 is active ($s_i \approx 1$), the dip characteristic of a VEB is included in the model and H4's activity should thus correspond to the existence of a VEB. In addition, the parameter a_i for H4 ends up with a large value after training, transforming H4 into a near-binary "decision maker", as indicated by Fig.4.14 and Fig.4.8(b). Therefore, the activities of hidden neurons, in particular that of H4, may form the basis of a simple novelty detector. The 1700 test data was presented one-by-one to the visible neurons of the trained CRBM. With the noise input of H4 removed ($\sigma = 0$), the post-sigmoid activities of H4 corresponding to the test data are shown in Fig.4.15. The strong peaks highlight the VEBs clearly. VL in Fig.4.15 indicates the minimum H4 activity for a VEB in the test data, and QH marks the maximum H4 activity induced by a normal QRS heartbeat. A simple linear classifier with

a threshold between the two dashed lines will therefore detect the VEBs with an accuracy of 100%. The margin for this threshold is more than 0.5, equivalent to 25% of the total value range. Compared to [69] where a 64-16-64 auto-associative MLP was trained to model the same data, the best accuracy achieved was merely 99.3%. Wang *et al.* also indicate that most neural-network models can only distinguish VEB from normal heartbeats with an accuracy of no more than 97% [70]. A single hidden neuron's activity in a CRBM is, therefore, potentially a reliable novelty detector and it is shown in [71,72] that layering a supervised classifier on the CRBM, to "fuse" the hidden neurons' activities, leads to improved results.

The parameter σ in Eq.(4.2) controls the noise variance of neurons in a CRBM. A neuron with a large value of σ becomes *binary-random*, as discussed in Sec.4.1.1, while a neuron with $\sigma = 0$ becomes *fully-deterministic*. To investigate the influence of the noise variance σ^2 on the CRBM's performance, CRBMs with various values of σ for the hidden neurons were trained to model the same ECG data. The margin allowed to detect the VEBs with 100% accuracy was then used to indicate the trained CRBMs' performance. Figure 4.16 shows the results of the CRBMs with two, four, and eight hidden neurons. The maximum margin always occurs with an intermediate value of σ. The degradation of the CRBM's performance with smaller σ can be attributed to 'overfitting' by *near-deterministic* hidden neurons, while the degradation with larger σ is more simply attributable to the domination of input noise. Figure 4.16 also reveals that increasing the number of hidden neurons does not necessarily improve the performance. A CRBM with eight hidden neurons "uses" more than one hidden neurons to model the characteristics of VEBs so the performance of each single hidden neuron as a VEB-detector is reduced. Clearly, a parsimonious architecture and an optimum level of noise are key to the CRBM's performance as a novelty, or rare-event detector.

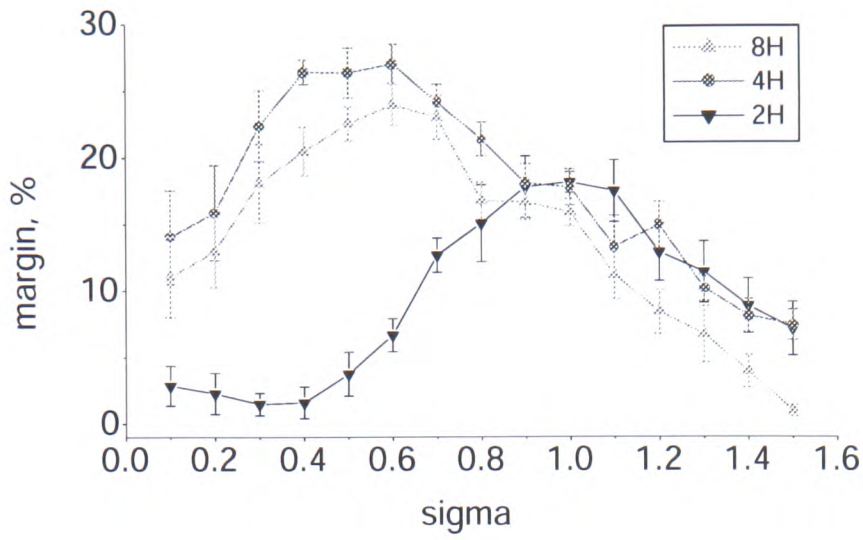


Figure 4.16: *Influence of noise variance σ on a CRBM's performance. The points and error bars show the mean and standard deviation across ten experiments.*

4.6 Summary

The CRBM is developed by introducing continuous stochastic neurons with Gaussian-noise inputs, and by inheriting the restricted architecture of the PoE/RBM. The development from the PoE/RBM, to the RBMrate, and then to the CRBM constitutes an evolution of "restricted" probabilistic models, from binary-valued to discrete-valued and then to continuous-valued. The simulation results in Sec.4.4 show that the CRBM is able to develop diverse stochastic behaviour in its neurons, in response to training, that range from binary-stochastic to deterministic. This leads to a modelling flexibility that exceeds the ability of the probabilistic models with fixed-form (e.g. binary) neurons. Experiments with real ECG data further demonstrate that the CRBM is capable of modelling complex continuous data.

The explicit form of the CRBM's equilibrium has not been derived. The lack of an explicit equilibrium form prevents the most principled method of classifying data, which is to calculate the log-likelihood of a test datum under a trained model. A low likelihood indicates that the corresponding test datum does not belong to the modelled category. However, the likelihood method requires complicated calculations, particularly in hardware, and multiple models are

necessary for modelling multiple categories. Likelihood calculation is thus impractical especially for an implantable intelligent system. Instead, it has been shown that the equilibrium of the CRBM approximates the continuous Boltzmann equilibrium when the conditions specified in Sec.4.2 are satisfied. This relationship justifies the derivation of the minimising-contrastive-divergence training rules for the CRBM, which are shown to be simple and hardware-friendly. Experiments further indicate that the training rules lead to intriguing and encouraging training behaviour. One CRBM is able to model multiple categories (or clusters) of data, and test data can be classified according to the responses of trained hidden neurons. Therefore, the CRBM provides a simple, pragmatic approach to classifying data, despite the lack of an exact, explicit form for its equilibrium distribution.

Alspector's works [21, 50] suggests the way to implement the continuous stochastic neurons of the CRBM, and the MCD training circuits has also been demonstrated in [52, 53], as described in Sec.2.8 and Sec.2.9. The CRBM is therefore a continuous probabilistic model that is both useful and amenable to VLSI implementation. In other words, the CRBM is a promising model for filling in the box with a question mark in Fig.2.17. To demonstrate the CRBM's hardware amenability and to examine the feasibility of realising VLSI circuits with continuous-stochastic behaviour, this research proceeds to implement the full CRBM model in VLSI with on-chip learning ability, as described in the following chapters.

Chapter 5
Mapping software simulation into VLSI implementation

The examples reviewed in Sec.2.8 and Sec.2.9 provide the basis for implementing both the stochastic neurons and the training rules of the CRBM in VLSI circuits. Prior to designing component circuits for the CRBM, a modular diagram is essential for specifying how component circuits will integrate to form a full CRBM system. In addition, it is important to take into account hardware limitations such as the voltage range and chip area available in hardware. Therefore, this chapter concerns possible simplifications for facilitating the VLSI implementation of the CRBM, and unavoidable hardware limitations which were not taken into account in the simulations in Ch.4. Matlab simulation is used to investigate whether the model retains reasonable performance under these simplifications and limitations. According to the simulation result, a modular diagram will subsequently be concluded for describing clearly how to "map" the CRBM algorithm into a CRBM system in VLSI.

5.1 Simplifications and limitations for hardware implementation

5.1.1 Simplified MCD training rules for the CRBM

Although the MCD training rules in Eq.(4.20) and Eq.(4.24) already possess a simple form, further simplifications are essential to facilitate their hardware implementation. Firstly, the values of the contrastive divergences[1] calculated in Eq.(4.20) and Eq.(4.24) vary from one

[1]The contrastive divergence here refers to the subtraction between two correlation terms in the training rules

training step to another. While updating parameters precisely by variable values is trivial in software simulation, it demands complicated circuits to achieve adequate precision in hardware. To reduce hardware complexity, the work reviewed in Sec.2.9 performs directional learning by taking only the sign of contrastive divergences. It is further shown that this simplification costs merely a longer training time without degrading the performance of the PoE/RBM [25]. Therefore, following this example, the MCD training rules of the CRBM are also simplified into directional training rules, so that parameters will be updated with fixed-size steps in hardware.

Secondly, the brackets $\langle\cdot\rangle$ in Eq.(4.20) and Eq.(4.24) denote the expectation value over all training data. If a training dataset contains 400 data points, the contrastive divergences suggested by 400 data are calculated, accumulated, and then averaged to obtain an expectation value in each training epoch. However, accumulating all these contrastive divergences is area-costly in hardware, as discussed in Sec.2.9. A compromise must be made between the number of accumulations and hardware area. It will be assumed that the expectation value over a subset of training data still provides a correct update direction most of the time, and that the effect of incorrect update directions can be minimised by reducing the step-size at each training step. Therefore, each expectation value is approximated by the average of only *four* training data, as opposed to that of all training data. The number of accumulations is chosen in an attempt to minimise area-cost without degrading the performance of the CRBM.

Let s_i and s_j denote the initial outputs of neuron i and neuron j, and let $\hat{s}_i$ and $\hat{s}_j$ denote the one-step Gibbs-sampled outputs of neuron i and neuron j. The simplified training rules are written as follows

$$\Delta\hat{w_{ij}} = \eta_w \cdot sgn\left(\langle s_i \cdot s_j\rangle_4 - \langle\hat{s}_i \cdot \hat{s}_j\rangle_4\right) \tag{5.1}$$

$$\Delta\hat{a}_i = \eta_a \cdot sgn\left(\langle s_i^2\rangle_4 - \langle\hat{s}_i{}^2\rangle_4\right) \tag{5.2}$$

where $sgn(\cdot)$ is the *signum* function with binary outputs of ± 1, and $\langle\cdot\rangle_4$ denotes taking average

over *four* training data. The denominator $1/a_i^2$ in Eq.(4.24) is excluded from Eq.(5.2), as $1/a_i^2$ is always positive and only the direction of contrastive divergence matters in the simplified training rules. Eq.(5.1) and Eq.(5.2) clearly indicates that contrastive divergences merely decide updating directions, and the learning rates η_w and η_a control the step-size at each training step. Sec.5.2 will use Matlab simulation to examine whether the CRBM remains capable of modelling data with these simplified training rules.

5.1.2 Limited value ranges of parameters

As parameters are designed to be represented as voltages stored on capacitors in hardware, the value ranges of parameters are limited by the power supply of VLSI circuits. The VLSI technology used in this research is the AMS 0.6 μm 2P3M CMOS process that operates with a supply voltage of 5V. All weight parameters $\{w_{ij}\}$ are thus confined to $[0, 5]$ (V), provided a wide-range multiplier is available for calculating $(w_{ij} \cdot s_j)$.

Ch.6 will show that each noise-control parameter a_i corresponds to a voltage that controls the feedback resistance of an I-V converter. The voltage range of $\{a_i\}$ thus has to be limited to $[1, 3]$(V), in order to ensure the normal operation of voltage-controlled active resistors [57]. As Sec.4.4 and Sec.4.5 show that the values of $\{a_i\}$ normally lie in $[0.5, 4]$ in all experiments, Ch.6 will design a voltage-control active resistor whose resistance varies at least by 0.5 to 4 times as its controlling voltage changes from 1 to 3 V.

In addition to $\{w_{ij}\}$ and $\{a_i\}$, neuron outputs $\{s_i\}$ will also be represented as voltages. The value range of $\{s_i\}$ is $[-1, 1]$ in simulation. The difference between maximum and minimum values equals 2. As a range of two volts is comfortably affordable in a 5-volt process, the voltage range of $\{s_i\}$ is not limited by the power supply. The mapping of $\{s_i\}$ from software simulation to hardware implementation can thus adopt a 1-to-1 mapping ratio.

	Matlab	VLSI (V)
s_i	$[-1.0, 1.0]$	$[1.5, 3.5]$
w_{ij}	$[-2.5, 2.5]$	$[0.0, 5.0]$
a_i	$[0.5, 4.0]$	$[1.0, 3.0]$

Table 5.1: *The mapping of parameter values between software simulation and hardware implementation.*

With the reference zero in hardware set at 2.5V, Table.5.1 summarises the mapping of parameter values between software simulation and hardware implementation. The mapping ratio for $\{w_{ij}\}$ is also chosen to be 1-to-1. The next section will examine the modelling capability of the CRBM when the values of its parameters are limited to the ranges specified in table.5.1.

5.2 Simulating a CRBM system in Matlab

To demonstrate the CRBM's hardware amenability and to explore the utility of continuous-valued probabilistic behaviour in VLSI, this research aims to implement a CRBM with two visible and four hidden neurons in VLSI. The number of hidden neurons was decided according to the simulation results in Sec.4.4, which indicate that four hidden neurons can model most types of two-dimensional data. To evaluate the modelling capability of such a CRBM system in VLSI, the CRBM system was simulated in Matlab with the simplified training rules in Eq.(5.1) and Eq.(5.2). The values of parameters were also limited to the ranges specified in table.5.1. The training data in Fig.5.1(a) was used to train the CRBM in simulation. The training data have the same underlying probability distribution (one elliptic and one circular Gaussian) as the training data in Fig.5.1(b). However, the total number of the training data is reduced to 20, in order to take into account the limited throughput of hardware-testing equipment. The reason for choosing this training data is because its particular distribution demands the CRBM's capability of not only regenerating a non-symmetric distribution but also modelling the probability

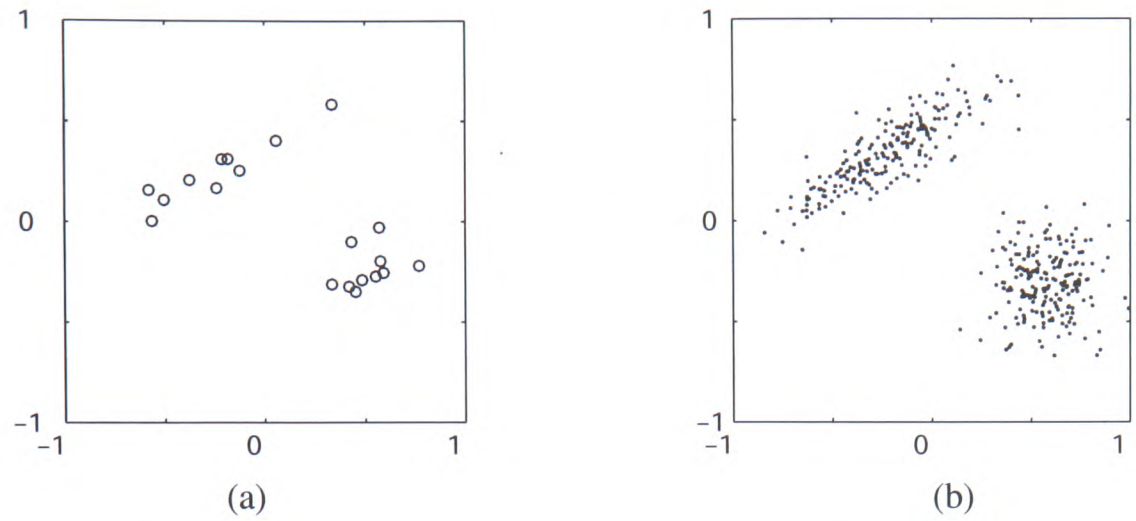


Figure 5.1: *(a)20 training data points sampled from a probability distribution of one elliptic and one circular Gaussians (b)The training data have the same distribution as that in (a), but contain 200 data points.*

distribution of training data correctly.

The CRBM was trained with $\eta_w = 0.003$, $\eta_a = 0.03$ for visible neurons, and $\eta_a = 0.003$ for hidden neurons. All learning rates were reduced to minimise the training errors introduced by the simplified training rules. The relatively higher learning rate for $\{a_i\}$ of the visible neurons was used to speed up the "autonomous annealing" process described in Sec.4.4.2. The initial values for $\{a_i\}$ of the visible neurons were also set to 3, in order to avoid the abrupt increase of $\{a_i\}$ at the onset of training. With the initial settings, the CRBM was trained for 30000 epochs. The training results are shown in Fig.5.2 to Fig.5.4, and are discussed as follows.

Fig.5.2 shows the evolution of parameters $\{a_i\}$ during training. The hard limits set to the range of $\{a_i\}$ are displayed clearly in Fig.5.2(a). With a relatively higher learning rate, the $\{a_i\}$ of visible neuron V1 rises towards 4 before epoch 10000, but is always "bounced back" by the hard limit. The $\{a_i\}$ of the visible neurons then anneal gradually towards smaller values, allowing the weights $\{w_{ij}\}$ to model the training data more detailedly. The lower hard limit again stops a_{V2} from becoming less than 0.5. The $\{a_i\}$ of most hidden neurons, on the contrary,

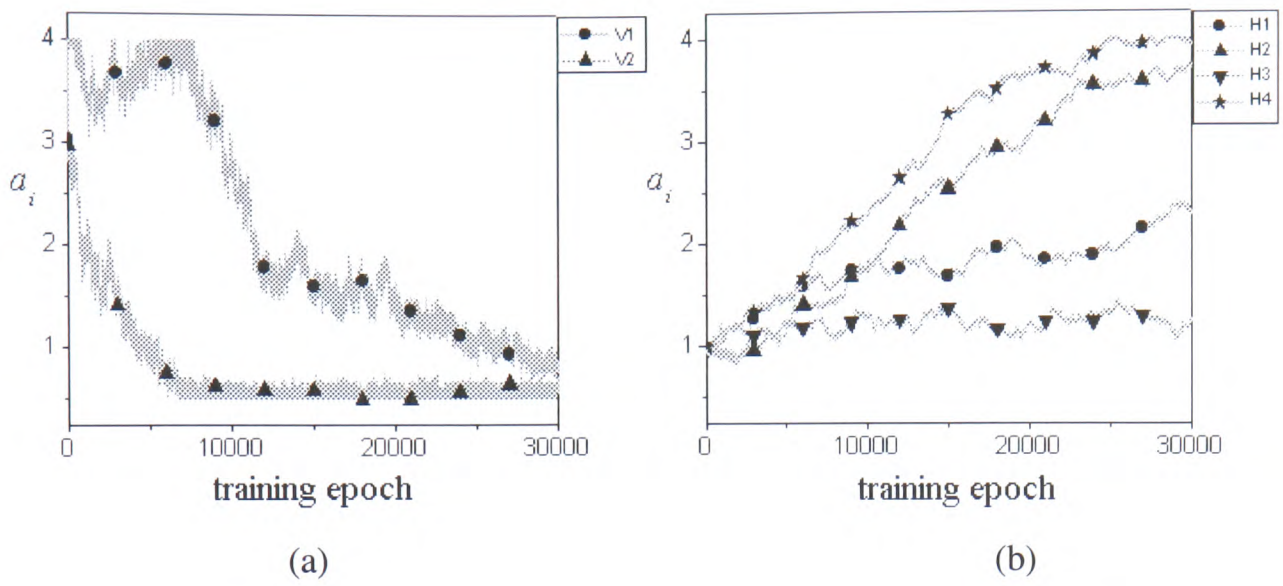


Figure 5.2: *The evolution of* $\{a_i\}$ *during 30000 training epochs (a)visible neurons (b) hidden neurons*

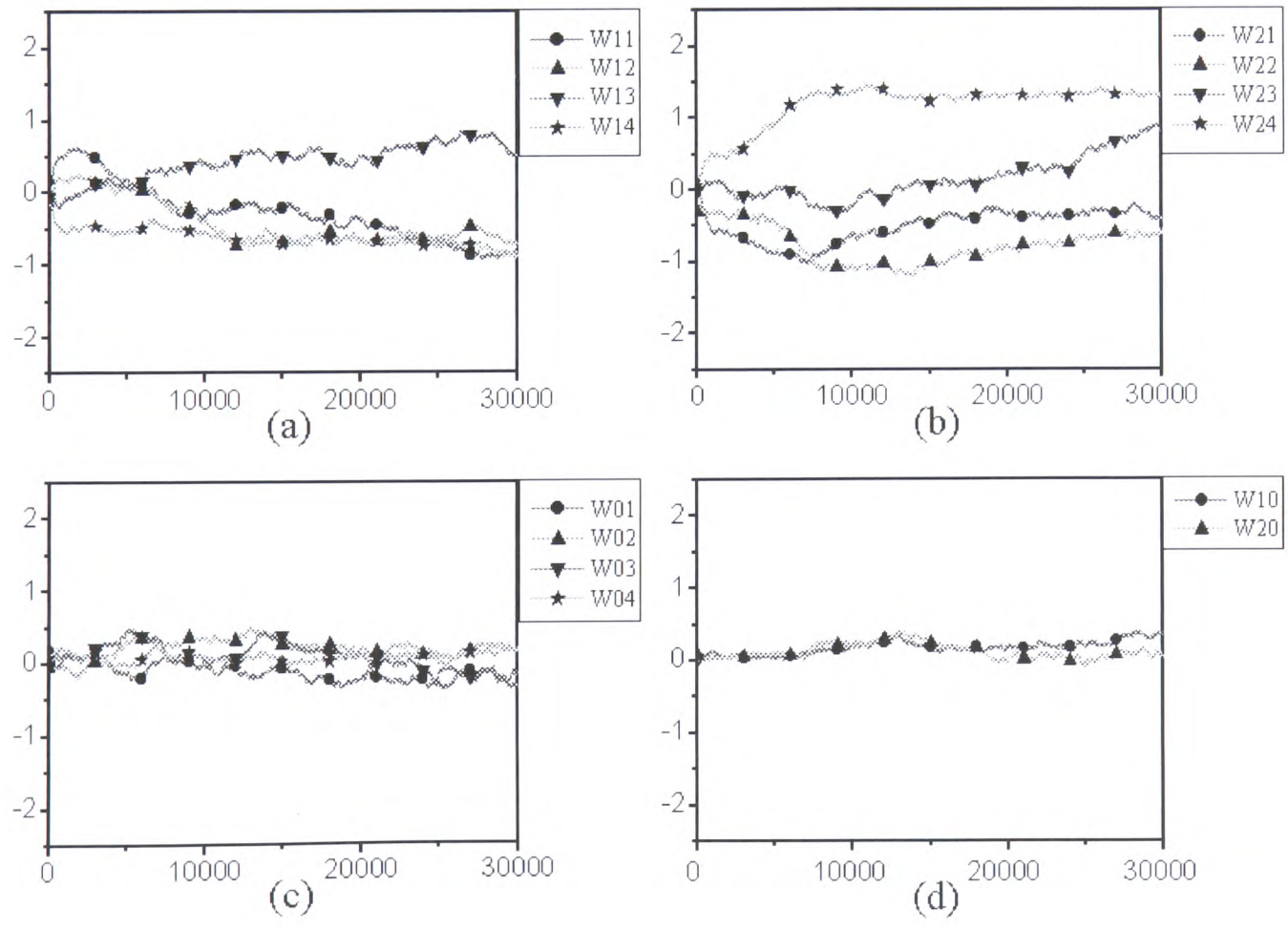


Figure 5.3: *The evolution of* $\{w_{ij}\}$ *during 30000 training epochs. (a)*$\{w_{1j}\}$ *(b)*$\{w_{2j}\}$ *(c)*$\{w_{0j}\}$ *(d)*$\{w_{i0}\}$.

stay in the limited range, as depicted in Fig.5.2(b). The only exception is that a_{H4} rises steadily to the maximum value of 4, because the knowledge vector of H4, though not shown here, learnt to encode the separation of the two training clusters. Fig.5.2 shows that all $\{a_i\}$ settle after 25000 epochs, indicating that the training process reaches equilibrium before epoch 30000.

Fig.5.3 shows the evolution of $\{w_{ij}\}$ during training. Each w_{ij} represents the connection between visible neuron i and hidden neuron j, and the index number for a bias unit is 0. Fig.5.3 shows that the values of all $\{w_{ij}\}$ lie in the limited range of $[-2.5, 2.5]$ comfortably, and that most $\{w_{ij}\}$ settle after 25000 epochs. As repeating the same experiment yields similar results, the experiment results imply that the value range designed for $\{w_{ij}\}$ in hardware is sufficient for the CRBM system to encode the distribution of two-dimensional data. Therefore, the mapping ratio for $\{w_{ij}\}$ is able to be set as 1-to-1 as in table.5.1.

After 30000 training epochs, the CRBM generated the 20-step reconstruction shown in Fig.5.4(a). The estimated statistical density assigned by the trained model is shown in Fig.5.4(b). Fig.5.4 reveals that the trained CRBM reconstructs a non-symmetric distribution successfully and models the underlying probability distribution of the training data. The promising results indicate that the simplifications made in Eq.(5.1) and Eq.(5.2) cause only a slower convergence time, while reducing the hardware complexity significantly. The next section will show each training epoch requires 24 clock cycles. Assume the CRBM system will operate at a frequency of 100kHz. Then 30000 training epochs correspond to a computing time of merely 7.2s in hardware. The slower convergence time is thus a small price to pay for simpler circuits and unavoidable hardware limitations.

The simple but nontrivial experiment in this section demonstrates that the CRBM remains capable of modelling two-dimensional training data faithfully, in spite of the simplifications and limitations applied to the model. The promising results encourage the VLSI implementation of

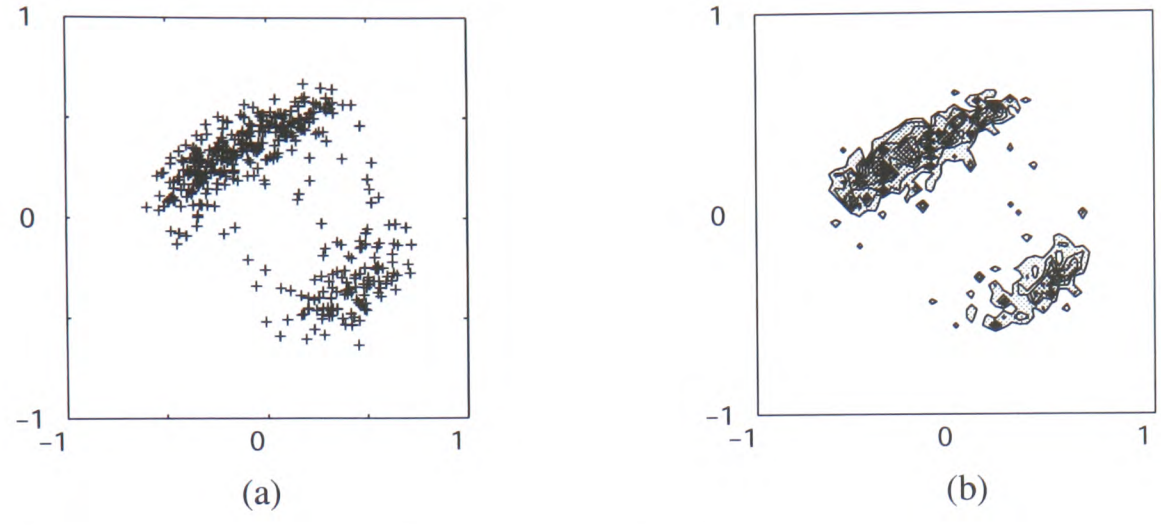


Figure 5.4: *(a)The 20-step reconstruction generated by the trained CRBM from 400 random initial data (b)The estimated statistical density over visible-state space assigned by the trained CRBM. The density is estimated by sampling 1000 reconstructed data from the trained model.*

a CRBM system according to Eq.(5.1), Eq.(5.2), and table.5.1. The next section will therefore derive the modular diagram of the CRBM system, as well as the digital control scheme for the system.

5.3 The modular diagram of a CRBM system

5.3.1 System architecture

Eq.(4.2) defines the CRBM neuron. The similarity between Eq.(4.2) and Eq.(2.37) indicates that the CRBM neuron is able to be implemented directly according to the block diagram in Fig.2.29, although several components in Fig.2.29 require an improved circuit design to enhance continuous-valued probabilistic computation. For a CRBM neuron, the voltage input V_{anneal} in Fig.2.29 corresponds to the parameter σ in Eq.(4.2) because both V_{anneal} and σ control the variance of input noise. As the noise-control parameter a_i of the CRBM adapts the slope of sigmoid function, a_i introduces the same effect to a neuron as the gain factor β in Eq.(2.37). The voltage input V_{gain} in Fig.2.29 thus corresponds to a_i of a CRBM neuron.

The simplified MCD training rules for the CRBM are given in Eq.(5.1) and Eq.(5.2). The two equations have exactly the same form as the simplified MCD training rule in Eq.(2.44). The MCD training rules for both $\{w_{ij}\}$ and $\{a_i\}$ can thus be implemented according to the same block diagram as shown in Fig.2.30. For training $\{w_{ij}\}$, the multiplier in Fig.2.30 calculates the correlation term $(s_i \cdot s_j)$, while for training $\{a_i\}$, the multiplier calculates the quadratic term s_i^2. In addition, the simulation in Sec.5.2 shows that four accumulations are enough for the CRBM to obtain a usable approximation to an expected contrastive divergence. The accumulator introduced in Sec.2.9.2 is thus suitable for the training circuits of the CRBM.

Based on the block diagrams in Fig.2.29 and Fig.2.30, the modular diagram of a full CRBM system is derived and shown in Fig.5.5. The CRBM system comprises six continuous stochastic neurons (two visible and four hidden neurons) and 20 slices of MCD training circuits that adapt all parameters of the neurons. The block diagram of each neuron is a simplified version of Fig.2.29, while the block diagram of each MCD training circuit is identical to Fig.2.30. The name of weight-changing circuits is changed to "learning circuit" to emphasize that not only weights $\{w_{ij}\}$ but also noise-control parameters $\{a_i\}$ are adapted in the CRBM system. The red-coloured pins in this figure denote the digital-control signals of the system. The blue-coloured pins denote the analogue signals transmitted between the CRBM system and its external environment, as well as the reference voltages and currents for the system. The yellow-coloured pins represent the noise inputs of neurons. All parameters are stored as voltages on capacitors, and are connected to the corresponding inputs of neurons, as indicated by the grey arrow at the bottom of the block diagram. The following paragraphs will describe the functions of individual blocks in Fig.5.5, while the circuit design will be detailed in Ch.6 and Ch.7.

The left half of the block diagram contains six continuous stochastic neurons. The two neurons on the top are visible neurons, while the four neurons on the bottom are hidden neurons. The multipliers calculate the deterministic input $\sum_j w_{ij} s_j$ in Eq.(4.2) for each neuron, and the

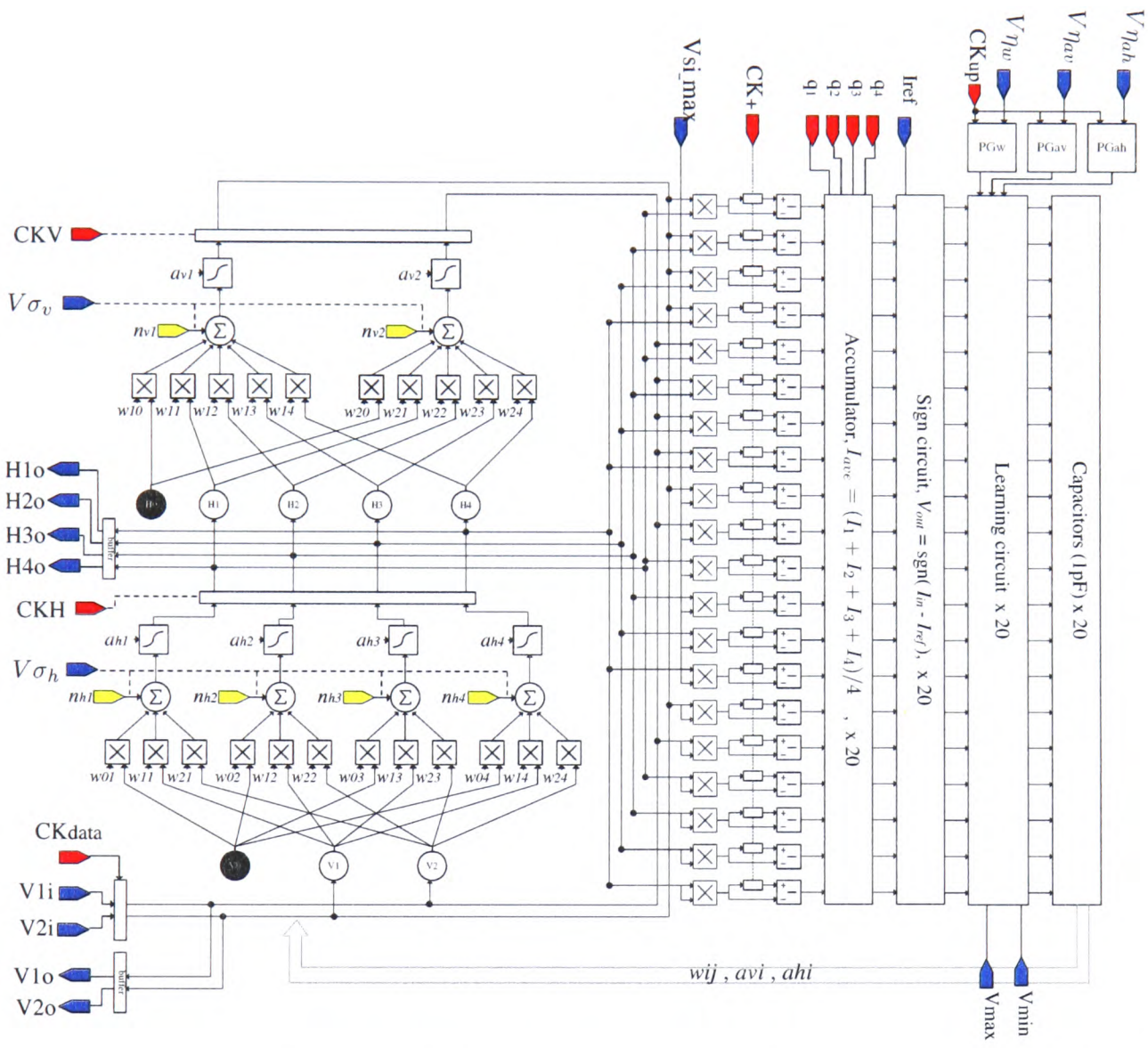


Figure 5.5: *The modular diagram of a full CRBM system with two visible and four hidden neurons. The two black circles denote bias units whose outputs always equal 1, and therefore are actually implemented by the voltage reference, Vsi_max, that defines* $s_i = 1$ *for a CRBM system.*

signals $V\sigma_v$ and $V\sigma_h$ control the variance of noise inputs, corresponding to the constant σ in Eq.(4.2). The summing circuits add the deterministic and noisy inputs, passing total inputs to the sigmoid-function circuits. As the noise inputs cause the sigmoid-function circuits to have continuous-valued probabilistic outputs, the control signals CKV and CKH are used to sample the outputs of visible and hidden neurons, respectively.

The right half of the block diagram contains the MCD training circuits. Each multiplier calculates either the correlation term $(s_i \cdot s_j)$ in Eq.(5.1) or the quadratic term s_i^2 in Eq.(5.2). After one-step Gibbs sampling is carried out in the neurons, the subtractors calculate the contrastive divergences between initial data and one-step Gibbs-sampled data. The accumulators then compute the average of four contrastive divergences, with the digital signals q1-q4 controlling the four accumulations. Subsequently, the sign circuits determine update directions for all parameters, and the learning circuits update all parameters once by charging or discharging the voltages stored on capacitors. Following the work in [52], the size of learning steps is designed to be controlled by pulse generators, as illustrated by the three square boxes above the training-circuit blocks in Fig.5.5. The inputs $V\eta_w$, $V\eta_{av}$, and $V\eta_{ah}$ control pulse widths and thus correspond to η_w for weights, η_a for visible neurons, and η_a for hidden neurons, respectively. Finally, the inputs V_{min} and V_{max} set the minimum and maximum voltages for $\{a_i\}$. Following the description of the modular diagram of a CRBM system, the next subsection presents the digital-control scheme for a CRBM system.

5.3.2 Digital control

The digital-control signals for a CRBM system during training are shown in Fig.5.6(a). CKref is the reference clock, setting the operation speed of the system. All other clock signals are synchronised with CKref, but at a lower frequency. To ease controlling a CRBM system and to simplify the design of digital-control circuits, only one clock signal is activated at each clock

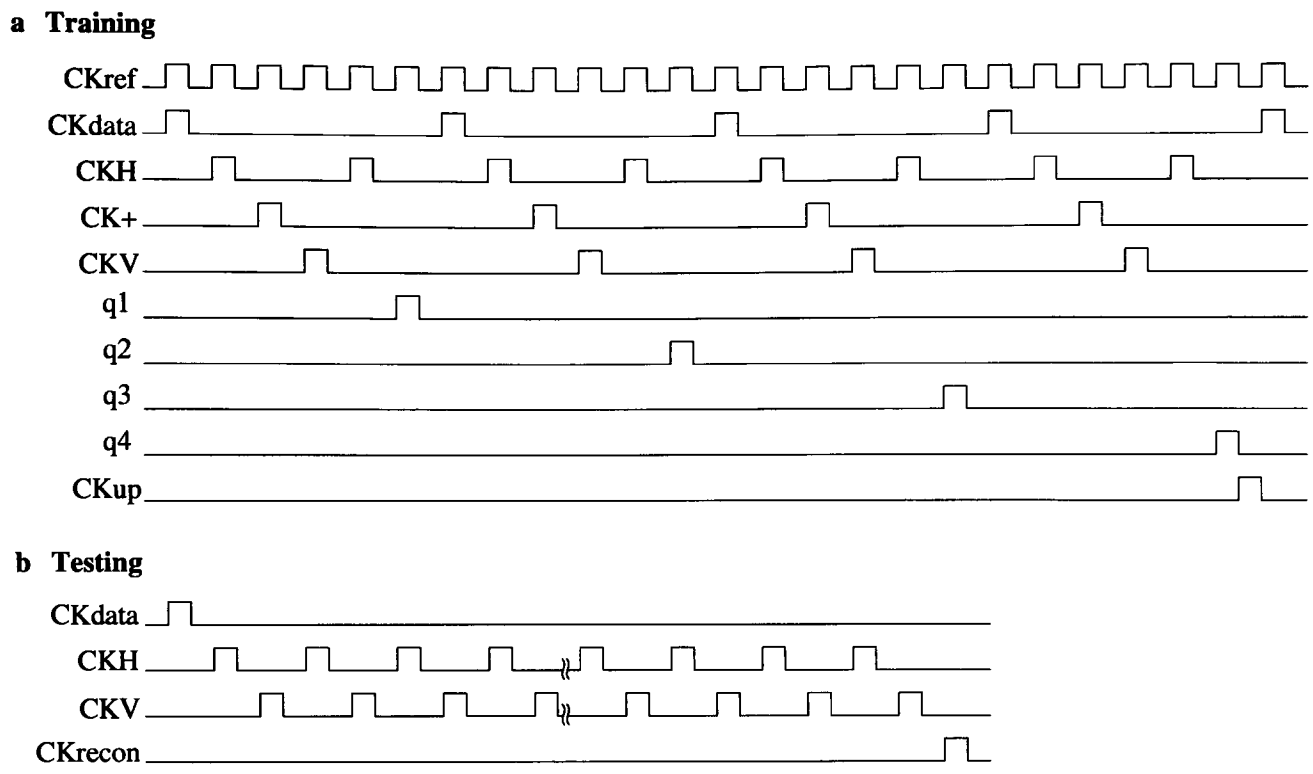


Figure 5.6: *The digital control signals for a CRBM system (a)during training (b)during multi-step reconstruction*

cycle. Initially, the CRBM system receives a training datum at the clock of CKdata. As the initial training datum acts as the initial state of visible neurons, the clock signal CKH samples the corresponding hidden state. The multipliers of training circuits then calculate the terms $(s_i \cdot s_j)$ and s_i^2 for the initial states. After CK+ samples and holds the values of $(s_i \cdot s_j)$ and s_i^2, CKV and CKH sample the visible and hidden neurons once more, obtaining the one-step Gibbs-sampled states of all neurons. q1 subsequently activates to sample and hold the contrastive divergences for the training datum. The same clock sequence is repeated three more times, and q2-q4 sample and hold the contrastive divergences suggested by another three training data. The clock signal CKup finally triggers the learning circuits to update all parameters once. Fig.5.6 indicates that updating parameters once takes 24 clock cycles. As an operating frequency faster than 100kHz can be achieved easily, the simulation results in Sec.5.2 imply that most training processes should be able to reach equilibrium in 10s.

Fig.5.6(b) shows the digital-control signals for the CRBM system during multi-step recon-

struction. The control signals are obviously simpler than those for a training process because only neurons are involved in a multi-step-reconstruction process. Initially, the CRBM system receives a random datum at the clock of CKdata. CKH and CKV then activate alternately to sample hidden and visible neurons for multiple steps. Finally, the nth-step reconstruction is read out from the pins V1o and V2o at the clock of CKrecon.

5.3.3 Discussion

The VLSI circuits for the individual blocks in Fig.5.5 have been suggested by the works reviewed in Sec.2.8 and Sec.2.9. The VLSI implementation of the CRBM system is thus feasible and simple. The CRBM system consists of noise-induced continuous-stochastic neurons, and therefore has continuous-valued probabilistic behaviour. MCD training circuits further enable the CRBM system to adapt its parameters on-chip. The CRBM system will therefore provide a platform for studying computation with noise-induced, continuous-valued probabilistic behaviour in VLSI.

Nevertheless, several components, as indicated by grey colours in Fig.2.29 and Fig.2.30, require improvements to enhance continuous-valued probabilistic computation in VLSI, while the use of an accumulator computing only four accumulations has been validated by the simulation in Sec.5.2. A voltage-limiter is also necessary for confining the voltage range of $\{a_i\}$. These component circuits will be designed and suggested in Ch.6 and Ch.7. Ch.8 will then gives a more detailed modular diagram that incorporates newly-designed components.

5.4 Summary

To facilitate the hardware implementation of the CRBM, the training rules of the CRBM are further simplified into Eq.(5.1) and Eq.(5.2). The values of parameters are also confined in the

ranges specified in table.5.1, in accord with voltage ranges available in hardware. After these simplifications and specifications are validated by Matlab simulation, the modular diagram of a full CRBM system together with its digital-control scheme is derived. The modular diagram in Fig.5.5 clearly illustrates the mapping of the CRBM algorithm into a CRBM system in VLSI, which has continuous-stochastic behaviour and on-chip adaptability. The CRBM system will thus provide a platform for exploring computation with continuous probabilistic behaviour in VLSI.

As the two main constituents of a CRBM system are continuous stochastic neurons and MCD training circuits, a CRBM system was implemented step-by-step as three VLSI chips. The first chip contains the VLSI implementation of continuous stochastic neurons, where the design of VLSI circuits with continuous-valued probabilistic behaviour is of concern. In the second chip, one slice of the MCD training circuit in Fig.5.5 was implemented, in order to achieve an efficient and reliable training circuit. The third chip finally integrates all component circuits according to Fig.5.5, constituting a full CRBM system on chip. The following three chapters present and discuss the design and measurements of the three chips respectively.

Chapter 6
Continuous-valued stochastic neuron in VLSI

This chapter concerns the VLSI implementation of continuous stochastic neurons for a CRBM system. As discussed in Sec.5.3, continuous stochastic neurons of the CRBM can be implemented according to the block diagram in Fig.2.29. Although the VLSI implementation of the Boltzmann Machine already suggests VLSI circuits for all blocks, several blocks require improved circuits to enhance continuous-valued probabilistic computation in VLSI. The circuits to be improved include a multiplier and a sigmoid-function circuit. This chapter will first describe the design and measurements of a multiplier and a sigmoid-function circuit that fit a CRBM system's requirement. The implementation of a noise generator that gives rise to probabilistic behaviour in VLSI will also be described. By combining all component circuits, a CRBM neuron is finally implemented in VLSI. The development and measurements of this VLSI neuron will be presented.

6.1 A wide-range, four-quadrant multiplier

Design considerations

According to the ranges for $\{s_i\}$ and $\{w_{ij}\}$ specified in table.5.1, a *wide-range, four-quadrant* multiplier is necessary for calculating $(w_{ij} \cdot s_i)$ at a neuron's input. The term *wide-range* refers to a wide input range for both $\{s_i\}$ and $\{w_{ij}\}$, within which the output of a multiplier retains linearly proportional to $(s_i \cdot w_{ij})$. The term *four-quadrant* then refers to the ability to calculate the multiplication of *signed* $\{s_i\}$ and $\{w_{ij}\}$. In addition, a simple architecture and a current-

mode output is preferable, as discussed in Sec.2.8.2. An analogue multiplier satisfying all these requirements is necessary.

The Gilbert multiplier [73] is a possible choice and has been employed in several neural systems [59, 73–75]. A MOS-version Gilbert multiplier exploits the *subthreshold* operation of input transistors to multiply voltages by adding their exponentially-related currents. However, the subthreshold operation requires a MOS transistor to have a gate-source voltage lower than its threshold voltage, whose absolute value is 0.7-0.9V in general. This requirement consequently limits the input range of a MOS-version Gilbert Multiplier to $\pm 500mV$ with respect to a reference-zero voltage, as depicted by the measurement results in [59, 74, 75]. Owing to this limitation, the Gilbert multiplier does not satisfy the need of a CRBM neuron, especially for $\{w_{ij}\}$ which demands a range of $[0, 5]$V. Although the multiplier suggested in [76] operates input transistors in the saturation region and extends its input range to $\pm 800mV$, the obtained range still does not fulfill a CRBM neuron's requirement. There are other reported multipliers achieving an input range of more than $\pm 1V$ [77–82]. However, these multipliers either require additional biasing circuits to produce differential-ended inputs [1] [77–79] or utilise particular biasing schemes to achieve a wide input range [80–82]. As pre-processing circuits and biasing schemes both complicate circuit architecture and make multipliers area- and power-consuming, these multipliers are unfavourable for the implementation of a CRBM neuron. Instead of adopting any reported multiplier directly, a "modified Chible multiplier" is therefore designed.

Design of a modified Chible multiplier

The Chible multiplier proposed in [83] has a simple architecture without a biasing circuit. Furthermore, one of the multiplier's inputs can have a range from rail(Gnd) to rail(V_{dd}) of a power supply. The only imperfection is that the reference zero of this input is process-dependent. Although the variations of reference zeros may ideally be compensated by training neurons'

[1] In circuits with differential-ended inputs or outputs, each voltage ΔV is represented as a pair of voltages $\left(V_{ref} + \frac{\Delta V}{2}, V_{ref} - \frac{\Delta V}{2}\right)$ where V_{ref} denotes a reference-zero voltage.

parameters, the lack of a unique reference zero discourages precise mapping of parameter values between a CRBM system and a CRBM model in Matlab simulation. In addition, the same four-quadrant multiplier could be used to implement the MCD training rules for the CRBM. A "modified Chible multiplier", as shown in Fig.6.1, is therefore designed to allow the reference zeros of both inputs to be controlled externally.

The modified Chible multiplier consists of two identical computing cells, each of which corresponds to a Chible multiplier proposed in [83]. The computing cell, as shown in Fig.6.1(a), contains two differential pairs biased by two complementary branches, Mn1-Mn2 and Mp1-Mp2. All transistors are biased in the *saturation* region [2]. Let n denote the *slope factor* of MOS transistors [3], and $V_{T,x}$ denote the absolute value of transistor Mx's *threshold voltage*. According to [83], the differential current $(I_{o1} - I_{o2})$ in Fig.6.1(a) is given as

$$I_{o1} - I_{o2} = \begin{cases} K_N(V_w - V_{MN}) \cdot (V_{si} - V_{sr}), & \text{if } V_w > V_{MN} \\ 0, & \text{if } V_w \leq V_{MN} \end{cases} \tag{6.1}$$

where K_N is a constant dependent on the size of the differential pair M1-M2, and $V_{MN} = (V_{T,n1} + nV_{T,n2})$. Eq.(6.1) indicates that V_w controls the bias current of the differential pair, such that the differential current $(I_{o1} - I_{o2})$ is proportional to V_w when $V_w > V_{MN}$. If $V_w \leq V_{MN}$, the current of the NMOS branch Mn1-Mn2 is turned off, and $(I_{o1} - I_{o2})$ consequently becomes zero. Similarly, the bias current of the differential pair M3-M4 is controlled by V_w, but is inversely proportional to V_w. The differential current $(I_{o3} - I_{o4})$ is given as [83]

$$I_{o3} - I_{o4} = \begin{cases} 0, & \text{if } V_w \geq V_{MP} \\ K_P(V_w - V_{MP}) \cdot (V_{si} - V_{sr}), & \text{if } V_w < V_{MP} \end{cases} \tag{6.2}$$

[2] A MOS transistor is biased in the *saturation* region when its gate-source voltage is greater than its threshold voltage (around 0.7-0.9V in a conventional VLSI technology).

[3] The slope factor n is a process-dependent constant having a value between 1 and 2 [84].

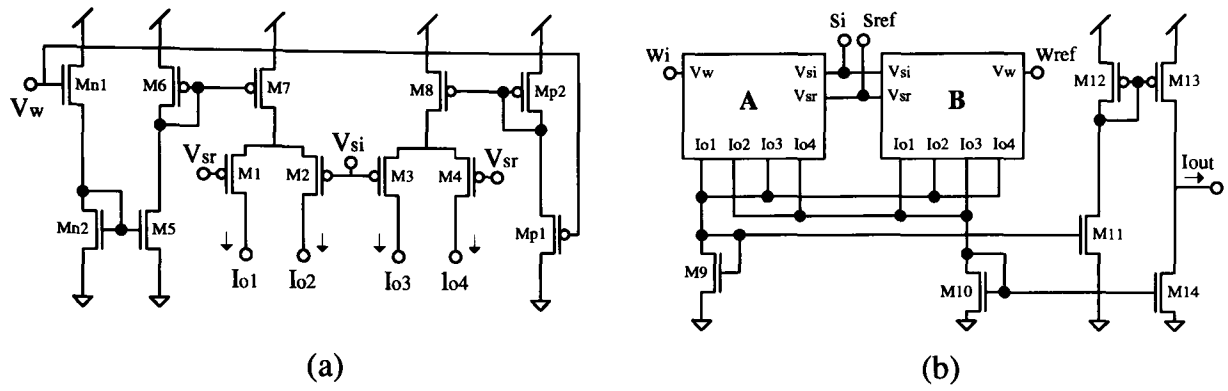


Figure 6.1: *The modified Chible four-quadrant multiplier (a)one computing cell of the modified Chible multiplier (b)the full circuit composed of two computing cells*

where K_P is a constant dependent on the size of the differential pair M3-M4, and $V_{MP} = (n^2 V_{dd} - V_{T,p1} - nV_{T,p2})$. V_{dd} represents the power-supply voltage. Combining Eq.(6.1) and Eq.(6.2) yields

$$I_o = (I_{o1} - I_{o2}) + (I_{o3} - I_{o4})$$
$$= \begin{cases} K_N(V_w - V_{MN}) \cdot (V_{si} - V_{sr}), & \text{if } V_w > V_{MN} \text{ and } V_w \geq V_{MP} \\ K_P(V_w - V_{MP}) \cdot (V_{si} - V_{sr}), & \text{if } V_w < V_{MP} \text{ and } V_w \leq V_{MN} \end{cases} \quad (6.3)$$

Subject to careful design of the complementary biasing transistors (Mn1-Mn2 and Mp1-Mp2), Chible claimed that $V_{MN} \approx V_{MP} \approx V_{dd}/2$ and thus that the cell is a four-quadrant multiplier [83]. However, threshold-voltage variations are normally greater than 10% in VLSI technology. This causes the reference zero for V_w (i.e. V_{MN} and V_{MP}) to vary from one multiplier to another significantly[4]. Another cell is therefore employed to cancel the effects of V_{MN} and V_{MP}, as shown in Fig.6.1(b). Let I_{oA} and I_{oB} represent the output currents calculated according to Eq.(6.3) for cell-A and cell-B, respectively. Transistors M9-M14 produce an output

[4]Simulation in Cadence shows that process variations cause the reference zero for V_w to vary from 2.288V to 2.830V

current equivalent to

$$
\begin{aligned}
I_{out} &= I_{oA} - I_{oB} \\
&= \begin{cases} K_N(W_i - W_{ref}) \cdot (S_i - S_{ref}), & \text{if } W_i > W_{ref} \\ K_P(W_i - W_{ref}) \cdot (S_i - S_{ref}), & \text{if } W_i < W_{ref} \end{cases}
\end{aligned} \tag{6.4}
$$

Eq.(6.4) shows that the subtraction between I_{oA} and I_{oB} cancels out V_{MN} and V_{MP} in Eq.(6.3). The reference zeros of W_i and S_i are thus set by external inputs, W_{ref} and S_{ref}. In addition, the range of W_i is that of the power supply. The modified Chible multiplier therefore provides a satisfactory input range for a CRBM neuron. Note that the incorporation of the second computing cell (cell B) does not expand power- and area- consumption significantly. As W_{ref} is set to $V_{dd}/2$ in cell B and $V_{MN} \approx V_{MP} \approx V_{dd}/2$, Eq.(6.1) and Eq.(6.2) indicates that cell B consumes a negligible current. The simple architecture of each computing cell, furthermore, renders the layout area of the modified Chible multiplier small [5].

Measurement result

Fig.6.2 shows the measured DC characteristics of a modified Chible Multiplier in a fabricated chip. The multiplier operates with a power-supply voltage of 5V, and the reference-zeros (W_{ref} and S_{ref}) are set to 2.5V. Each curve in Fig.6.2 corresponds to a particular voltage of S_i, with W_i sweeping from 0 to 5 V. Clearly, the multiplier exhibits an approximately-linear input-output relationship over the desired input ranges for $\{w_i\}$ and $\{s_i\}$ in table.5.1. The small nonlinearity displayed in Fig.6.2 should not affect the performance of a CRBM neuron, as a CRBM neuron adapts $\{w_{ij}\}$ to obtain its desired current inputs. The modified Chible multiplier therefore suits the requirement of a CRBM neuron, for calculating the term $\sum_j w_{ij}s_j$ in Eq.(4.2). In addition, Fig.6.2 reveals that $I_{out} = 0$ occurs at precisely either $W_i = 2.5V$ or $S_i = 2.5V$. This multiplier is also suitable for implementing the MCD training rules for the CRBM.

[5] In the implementation without minimising layout area of circuits, the layout area of a modified Chible multiplier is $126\mu m \times 98\mu m$

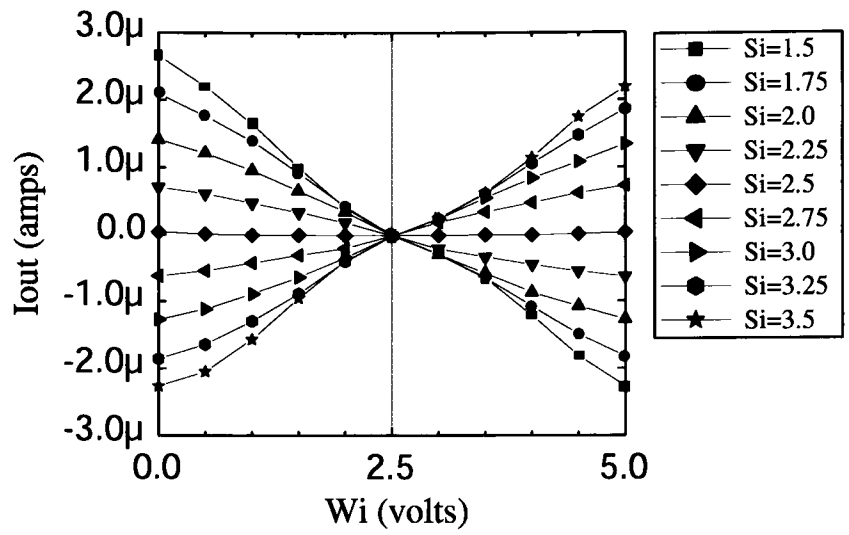


Figure 6.2: *The measured DC characteristics of the modified Chible multiplier*

6.2 Sigmoidal-function circuit

Design considerations

The sigmoid function of a CRBM neuron is defined in Eq.(4.3). Parameter a_i controls the slope of the sigmoid function and thus the influence of noise, as illustrated by Fig.4.8. Although the example work in Sec.2.8 suggests a sigmoid-function circuit, the suggested circuit alters the asymptotes of the sigmoidal nonlinearity when adapting the slope of the sigmoidal nonlinearity, as shown in Fig.2.23. Such alteration limits neurons' output range significantly, and may consequently degrade the modelling capability of the CRBM. To enhance continuous-valued probabilistic computation, a sigmoid-function circuit that is able to vary its slope without altering its output asymptotes is preferred.

The sigmoid-function circuits reported in [85–88] meet this requirement, but still do not meet the requirement of a CRBM neuron. The circuits in [85, 86] are based upon an input differential pair of MOS transistors (Fig.2.21), and bias transistors in the *saturation* region. As the current-voltage relationship of a MOS in saturation follows a square law rather than an exponential law [55], the exponential characteristics of a sigmoidal nonlinearity are not faithfully implemented in [85, 86]. As non-faithful implementation potentially reduces the modelling

richness of a continuous-valued model, the circuits in [85,86] are not ideal for the implementation of a CRBM neuron. The circuits in [87,88] bias input differential pairs in the *subthreshold* region to realise a sigmoidal nonlinearity as faithful as possible. However, threshold-voltage variations are normally as large as 10%, corresponding to a variation of around $\pm 100mV$ from a mean value. Such a non-negligible variation makes it difficult to ensure the subthreshold operation of MOS transistors for all neurons, especially for CRBM neurons whose inputs are expected to be "noisy". Therefore, sigmoid-function circuits for CRBM neurons are implemented by exploiting the *lateral-bipolar* operation of MOS transistors [89], which is characterised by an exponential current-voltage relationship over a wide input range.

Design of a sigmoid-fimction circuit basing on lateral-bipolar operation

Fig.6.3(a) illustrates a PMOS transistor fabricated in an n-well process. The lateral bipolar operation of the PMOS transistor refers to biasing its p-type source, n-well bulk, and p-type drain as a bipolar transistor ($V_{SB} > V_{SD} > 0$), such that a bipolar mode of current conduction is induced under the gate and in the lateral direction (as indicated by the grey arrow in Fig.6.3(a)) [89]. The gate of the MOS transistor is then connected to the most positive (for PMOS) or negative potential (for NMOS) to induce an accumulation layer of majority carriers under the gate [89]. Fig.6.3(b) illustrates the symbol denoting a PMOS transistor in the lateral-bipolar mode (called "lateral-bipolar transistor" in the following). In the lateral-bipolar mode, the source, bulk, and drain of a MOS transistor function as the emitter, base, collector of a bipolar transistor, respectively. The current-voltage relationship between I_d and V_{sb} follows [89]

$$I_d = Ke^{\frac{V_{sb}}{U_T}} \tag{6.5}$$

where K is a constant proportional to the effective area of the base-emitter junction, and U_T is the *thermal voltage* approximately equivalent to $25mV$. Eq.(6.5) indicates clearly that lateral-bipolar transistors introduce an exponential relationship, analogous to that in Eq.(2.39) intro-

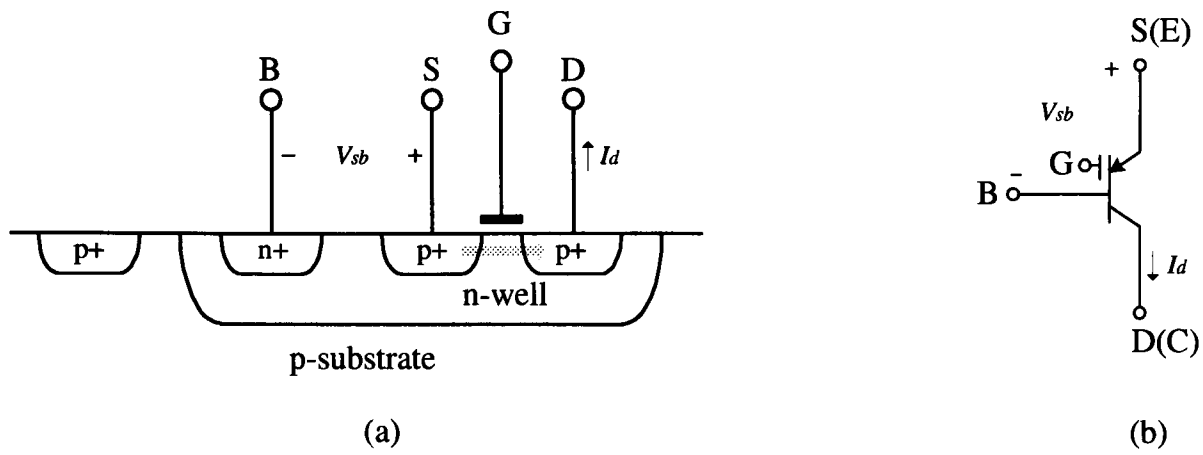


Figure 6.3: *(a)A PMOS transistor fabricated in the n-well process. (b)The symbol for the PMOS transistor operated in the lateral-bipolar mode*

duced by the subthreshold operation. Furthermore, this exponential relationship holds for any level of V_{sb}, as it comes from bipolar conduction.

Following [90], the sigmoid-function circuit is implemented by combining a differential pair of lateral-bipolar transistors with an I-V converter, as shown in Fig.6.4(a). The input current I_{in} represents the total input current to a neuron, including the input of a noise current. With a voltage-controlled feedback resistor [57], the I-V converter converts I_{in} into

$$V_x = V_{ref} - I_{in} \cdot R(V_{ai}) \tag{6.6}$$

where $R(V_{ai})$ indicates that the feedback resistance is controlled by the input voltage V_{ai}. The differential pair of lateral-bipolar transistors, Mbp1-Mbp2, then introduces the sigmoidal non-linearity between V_x and $(i_{c1} - i_{c2})$. According to Eq.(6.5), the differential current $(i_{c1} - i_{c2})$ is

$$\begin{aligned} i_{c1} - i_{c2} &= I_b \cdot tanh\left(\frac{V_{ref} - V_x}{2U_T}\right) \\ &= I_b \cdot \phi\left(\frac{I_{in} \cdot R(V_{ai})}{U_T}\right) \end{aligned} \tag{6.7}$$

where $\phi(\cdot)$ denotes the sigmoid function $\varphi_i(\cdot)$ in Eq.(4.3) with $\theta_H = -\theta_L = 1$ and $a_i = 1$.

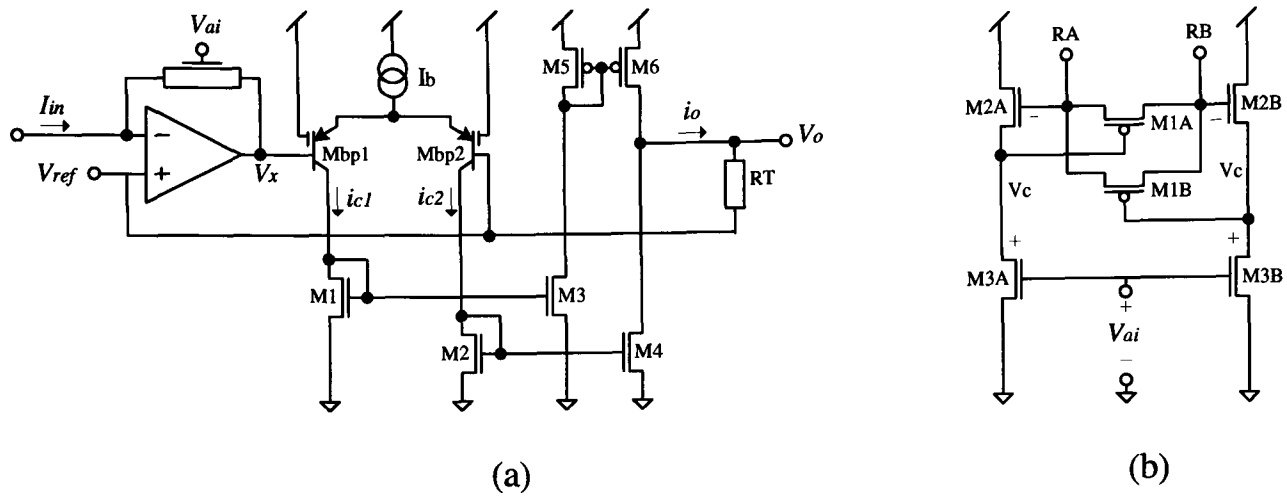


Figure 6.4: *(a)The sigmoid-function circuit for a CRBM neuron (b)The circuit diagram of the voltage-controlled feedback resistor in (a)*

Finally, transistors M1-M6 integrate i_{c1} and i_{c2} into a single output current $i_o = (i_{c1} - i_{c2})$, and the resistor RT transforms i_o into an output voltage of

$$\begin{aligned} V_o &= i_o R_T + V_{ref} \\ &= I_b R_T \cdot \phi\left(\frac{I_{in} \cdot R(V_{ai})}{U_T}\right) + V_{ref} \end{aligned} \tag{6.8}$$

Eq.(6.8) indicates that the relationship between V_o and I_{in} is exactly the sigmoidal nonlinearity of a CRBM neuron. In addition, Eq.(6.8) implies that V_{ai} adapts the slope of the sigmoidal nonlinearity by controlling the feedback resistance $R(V_{ai})$. The input voltage V_{ai} thus corresponds to the noise-control parameter a_i in Eq.(4.3), as will be shown in measurement results.

Fig.6.4(b) shows the circuit diagram of the voltage-controlled active resistor used in the sigmoid-function circuit in Fig.6.4(a). Transistors M1A, M2A, and M3A are identical to M1B, M2B, and M3B, respectively. The input voltage V_{ai} controls the current flowing through the two branches of NMOS transistors, and thus determines the gate-source voltages of M2A and M2B, which is denoted as V_c in Fig.6.4(b). V_c then determines the sum of the drain-source conductances of M1A and M1B, i.e. the inverse of the resistance between RA and RB, according

to [57]

$$G(V_c) = \frac{1}{R(V_c)} = 2\beta_1(V_c - V_{T,1}) \tag{6.9}$$

where β_1 is the transconductance [6] of M1A and M1B, and $V_{T,1}$ the threshold voltage of M1A and M1B. As V_c increases with V_{ai}, Eq.(6.9) indicates that the resistance is inversely proportional to V_{ai}.

Note that V_{ai} merely controls the resistance effectively in a limited voltage range. Although V_c and conductance $G(V_c)$ increases with V_{ai}, the maximum attainable value for V_c is the voltage difference between RA (or RB) and the ground. Beyond this value, M3A and M3B are switched off and a further increase of V_{ai} no longer induces a higher conductance. Moreover, if a small value of V_{ai} results in a V_c smaller than the threshold voltages of M1A and M1B, M1A and M1B enter the subthreshold region and, as a consequence, the linear relationship in Eq.(6.9) no longer holds. Owing to the consideration above, the voltage range of V_{ai} is designed to be confined within $[1, 3]$(V). Let the minimum resistance obtained at $V_{ai} = 3V$ correspond to $a_i = 0.5$ in Matlab simulation. To meet the range $[0.5, 4]$ specified for a_i, the resistance $R(V_{ai})$ is designed to be eight times its minimum value when V_{ai} decreases from 3 to 1 volts.

Measurement result

To examine the sigmoidal nonlinearity introduced by the differential pair of lateral-bipolar transistors, the DC characteristics of V_o in response to variable V_x in Fig.6.4 are measured. The black solid curve in Fig.6.5(a) depicts the measured V_o when $(V_{ref} - V_x)$ varies from -250mV to 250mV. For comparison, the grey dashed curve displays an ideal sigmoidal nonlinearity, $\phi(x)$, calculated according to Eq.(4.3). In x-axis, $(V_{ref} - V_x)$ is normalised by $U_T = 25mV$ to ease comparison. Fig.6.5(a) shows that V_o follows a sigmoid-shaped curve from 1.5V to 3.5V when $(V_{ref} - V_x)$ sweeps from -250mV to 250mV. In addition, the measured curve and the

[6] $\beta = \mu C_{ox} W/L$ denotes the transconductance of a transistor. W/L is the width-to-length ratio of the transistor, μ is the effective mobility, and C_{ox} is the gate-oxide capacitance per unit area.

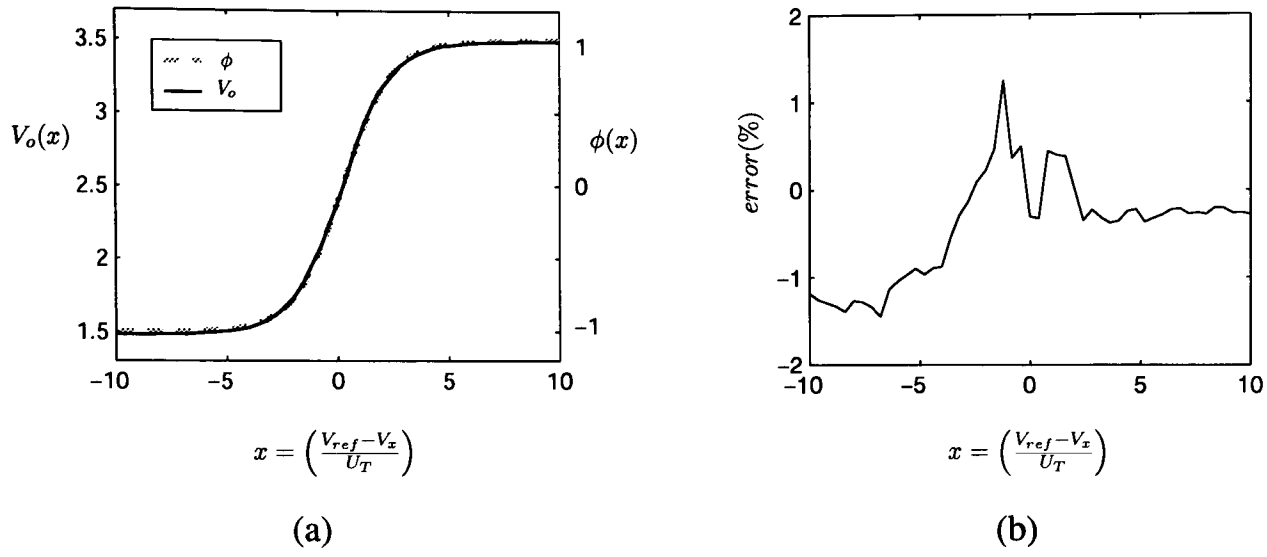


Figure 6.5: *(a)The measured DC characteristics of the differential pair of lateral-bipolar transistors (solid curve). The dashed curve is the ideal sigmoidal nonlinearity defined by $\phi(\cdot)$. (b)The error between the measured curve and the ideal curve.*

ideal curve are eventually indistinguishable. The errors between the two curves are shown in Fig.6.5(b), and are less than 1.5% over the measured range. This result indicates that lateral-bipolar transistors introduce a satisfactory sigmoid.

Fig.6.6 shows the measured DC characteristics of the full sigmoid-function circuit. Each curve corresponds to a measured DC relationship between V_o and I_{in} when V_{ai} is set to a particular potential level. Fig.6.6 shows that V_{ai} adapts the slope of the sigmoidal nonlinearity effectively as parameter $\{a_i\}$ in Eq.(4.3), without altering the asymptotes of the sigmoidal nonlinearity. To quantify the effect of V_{ai}, a mapping between V_{ai} and $\{a_i\}$ in simulation is derived by the following steps.

1. Select $V_{ai} = 1.8V$ to correspond to $a_i = 1$ in simulation.

2. Choose a reference current input I_r. Let V_r denote the output voltage of the sigmoid-function circuit when $I_{in} = I_r$ and $V_{ai} = 1.8V$

3. With the same input current I_r but various levels of V_{ai}, measure the output voltages of the sigmoid-function circuits, $V_o(V_{ai})$.

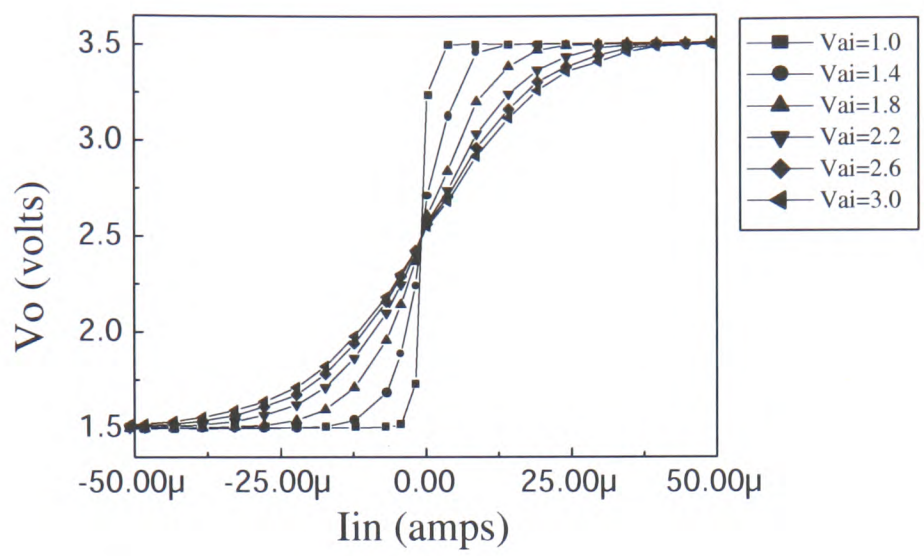


Figure 6.6: *The measured DC characteristics of the sigmoid-function circuit*

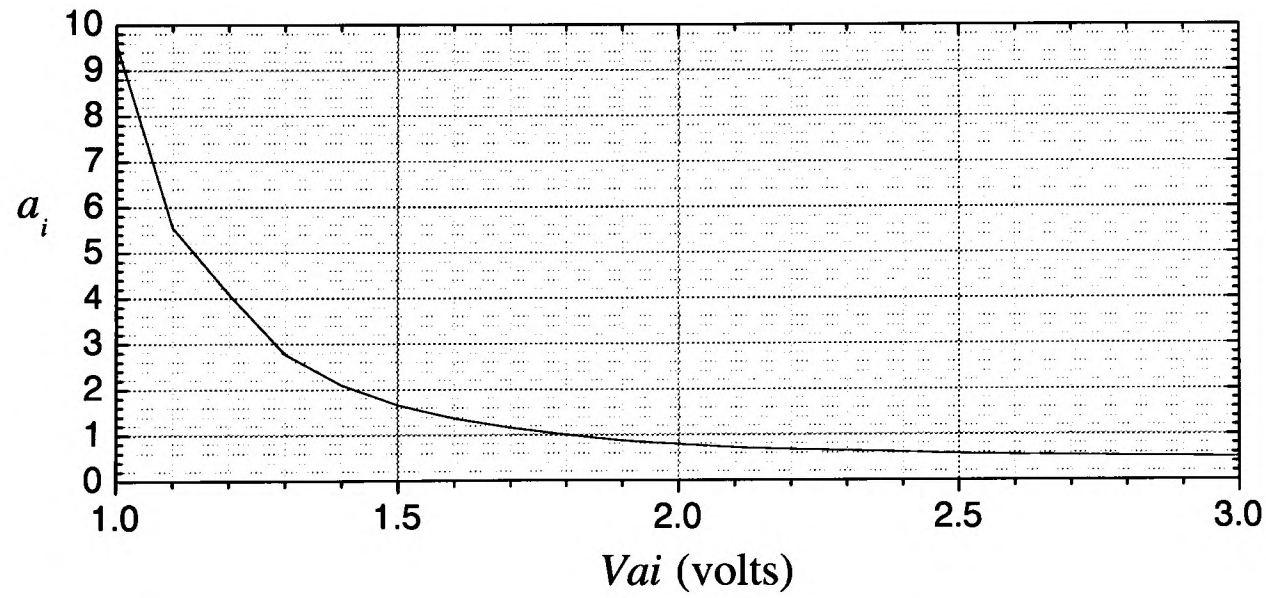


Figure 6.7: *The mapping between hardware implementation and software simulation for parameter* $\{a_i\}$.

4. The mapping between V_{ai} and a_i is then derived according to

$$a_i(V_{ai}) = \frac{\phi^{-1}\left(V_o(V_{ai})\right)}{\phi^{-1}\left(V_r\right)} \tag{6.10}$$

where $\phi^{-1}(\cdot)$ is the inverse of the sigmoid function $\phi(\cdot)$ Fig.6.7 shows the derived mapping between V_{ai} and $\{a_i\}$. The result is an average over measurements from three fabricated chips. Fig.6.7 reveals that a voltage range of $[1,3]$V for V_{ai} in hardware corresponds to a value range of $[0.5,9.5]$ for $\{a_i\}$ in simulation. This range covers the desired range, $[0.5,4]$, for $\{a_i\}$ in table.5.1. The sigmoid function for a CRBM neuron (defined in Eq.(4.3)) is therefore implemented successfully.

Furthermore, the mapping curve in Fig.6.7 facilitates the comparison between Matlab simulation and hardware implementation of the CRBM. While a 1-to-1 ratio is applicable to map $\{w_{ij}\}$ between hardware implementation and software simulation, the curve in Fig.6.7 details the mapping for $\{a_i\}$. Parameters $\{w_{ij}\}$ and $\{a_i\}$ learnt in Matlab are thus able to be "downloaded" onto a CRBM system, for testing the reconstructing (recalling) phase of the CRBM system. Or inversely, parameters learnt by a CRBM system are able to be "uploaded" into Matlab for inspecting learning results in hardware. This will be shown to be useful and important in Ch.8.

Now that a multiplier and a sigmoid-function circuit fitting the requirement of a CRBM system have been designed, a CRBM neuron is ready to be implemented in VLSI. Nevertheless, before describing the VLSI implementation of a CRBM neuron, the following section presents the design and measurement results of a noise generator that supplies Gaussian noise sources to CRBM neurons.

channel number	flip-flop number			Shift (bits)
1	4	12	16	813
2	1	10	19	3,065,848
3	11	18	25	6,561,452
4	11	12	18	9,153,942
5	2	9	19	12,240,651
6	3	12	22	15,641,663
7	4	6	23	18,739,334
8	5	8	24	22,115,686
9	7	14	18	25,165,828
10	2	12	16	28,164,123

Table 6.1: *Tap-pattern table for generating 10 channels of pseudo-random bit streams from a 25-stage LFSR. Three flip-flop numbers in each row specify the flip-flops (numbered from 1 to 25) to be tapped to a 3-input-XOR gate.*

6.3 Noise generator

Design of a LFSR-based noise generator

According to the exemplary work introduced in Sec.2.8, a noise generator based on a 25-stage LFSR (linear-feedback-shift-register) has been implemented. The outputs of 25 flip-flops in the LFSR are connected to ten 3-input-XOR gates, to generate ten channels of pseudo-random bit streams. The tapping pattern between flip-flops and XOR gates is specified in table.6.1. As each 3-input-XOR gate outputs a shifted version of a pseudo-random bit stream, the right hand column in table.6.1 gives shifting separations between bit streams in terms of shifted bits [21]. These ten pseudo-random bit streams are then transformed into ten analogue noise sources by simple RC filters. To achieve a good approximation to Gaussian noise, the 3-dB frequency of RC filters is designed to be 100 times smaller than the LFSR clock frequency (100MHz).

Table.6.1 indicates that the minimum separation between two channels of pseudo-random bit streams is 2,592,490 bits (between ch3 and ch4). According to Eq.(2.42), any two channels

of analogue noise sources are uncorrelated for more than

$$t_u = \frac{2,592,490bits}{100MHz} \approx 25.9ms \tag{6.11}$$

For training a CRBM system, t_u has to be longer than the time required for one step of Gibbs sampling between neurons. The digital-control scheme in Fig.5.6 shows that one-step Gibbs sampling takes six clock cycles. As a CRBM system is designed to be clocked at 100kHz, six clock cycles correspond to only 60 μs in time. The uncorrelated period, t_u, between analogue noise sources is therefore sufficient for training a CRBM system, even if the CRBM system is clocked at a frequency as low as 1kHz.

Measurement result

Fig.6.8(a) shows one analogue noise signal measured from a fabricated LFSR noise generator. The signal level is shifted to a mean of around 0V by subtracting 2.5V. As an oscilloscope with a sampling rate of 200M samples/s was used, the noise signal in Fig.6.8(a) contains 40000 samples over 200us. The amplitude distribution of the 40000 samples is illustrated in Fig.6.8(b). The amplitude distribution approximates a Gaussian distribution, and the grey curve is a Gaussian fit to the amplitude distribution. The mean (μ) and the width (2σ) are -0.167 and 0.9, respectively. As this indicates that 95% of noise amplitudes lie in $[-1, 1]$(V), i.e. $[1.5, 3.5]$(V) for un-shifted raw signals, the generated noise signal can simply be fed into a neuron through a differential pair, as illustrated in Fig.2.22. Measurements with other channels of analogue noise sources yield similar results. The LFSR noise generator is thus suitable for generating near-Gaussian noise for CRBM neurons.

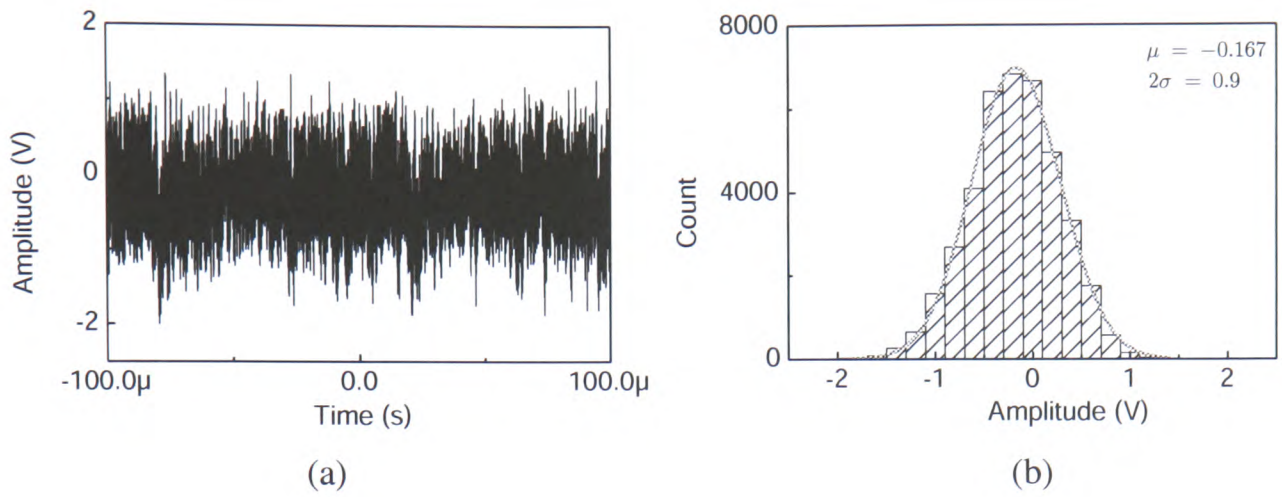


Figure 6.8: *(a)The measured signal from one channel of a LFSR noise generator (b)The amplitude distribution of the measured noise signal.*

6.4 A CRBM neuron in VLSI

Circuit architecture

Combining all component circuits introduced in previous sections, a CRBM neuron in VLSI is realised as described by Eq.(4.2). Fig.6.9 shows the CRBM neuron in VLSI. The position of the noise input V_{noise} differs from that in the block diagram in Fig.2.29. The noise input precedes the I-V converter through the differential pair, Mp1-Mp2, while a noise input is included after an I-V converter in Fig.2.29. The current i_{in} in Fig.6.9 thus represents the total input to a neuron, instead of deterministic inputs only. The wide-range four-quadrant multipliers output a deterministic current , i_{sum}, proportional to $\sum_j w_{ij}s_j$. The differential pair of Mp1 and Mp2 then converts V_{noise} into a noise current $i_n = g_m(V_{noise} - V_{nref})$, where V_σ controls g_m and thus scales the noise current as σ in Eq.(4.2). As a result, i_{in} represents the pre-sigmoid input $(\sum_j w_{ij}s_j + \sigma \cdot N(0,1))$ in Eq.(4.2). The sigmoid-function circuit subsequently introduces a sigmoidal relationship between i_{in} and v_o, giving rise to a v_o which represents s_i in Eq.(4.2). Finally, the switch is used to sample an output value, V_{si}. Results of a CRBM neuron in VLSI are presented and discussed in the following section.

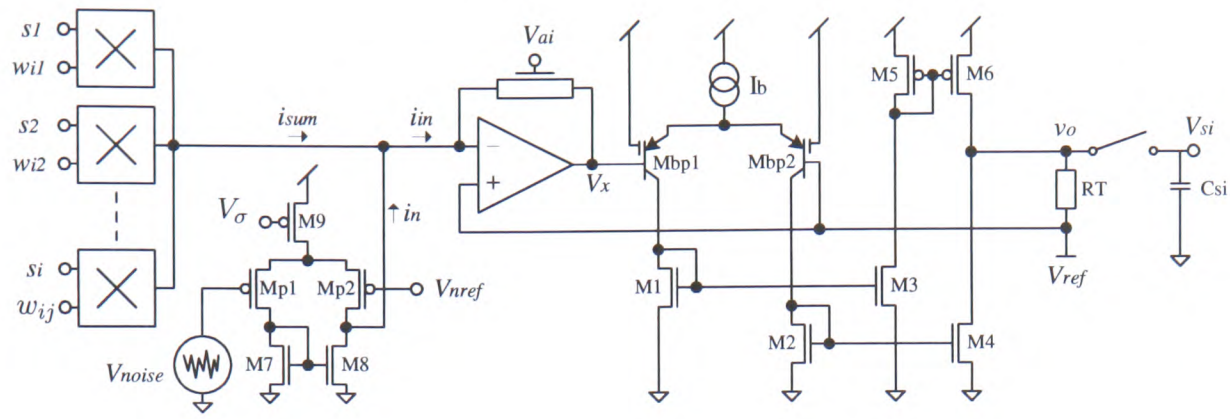


Figure 6.9: *The circuit diagram of a CRBM neuron in VLSI.*

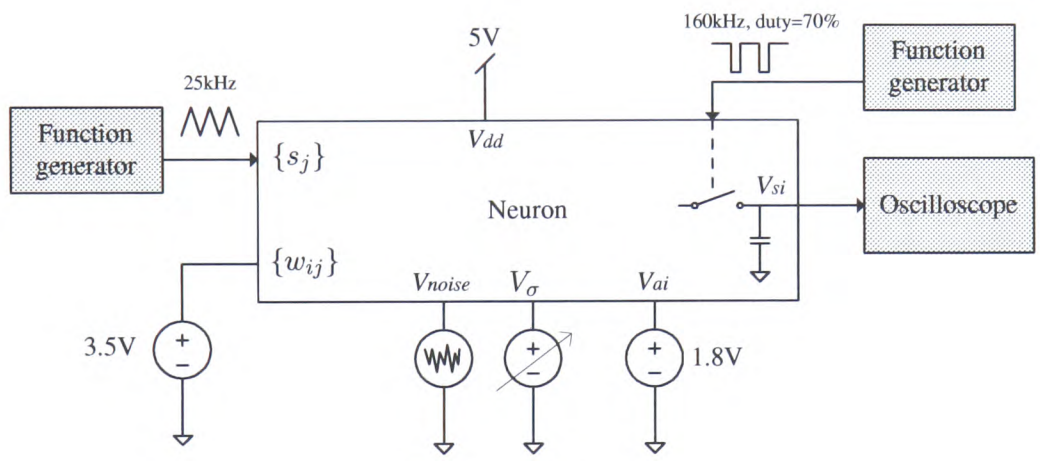
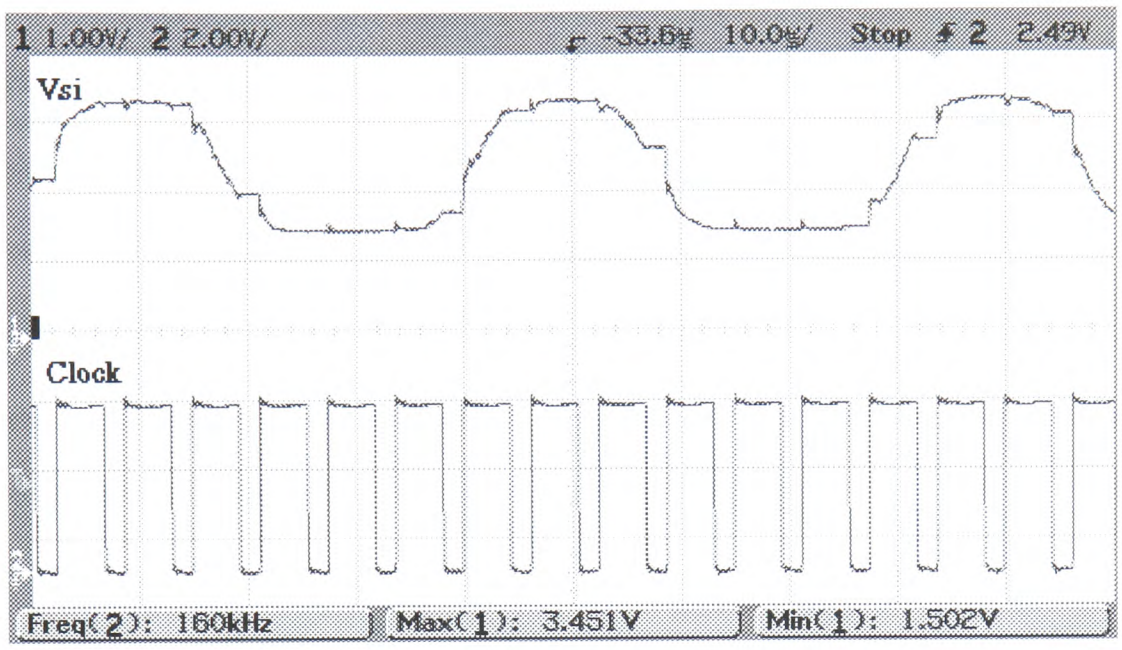


Figure 6.10: *The measurement setup for testing a CRBM neuron in VLSI.*

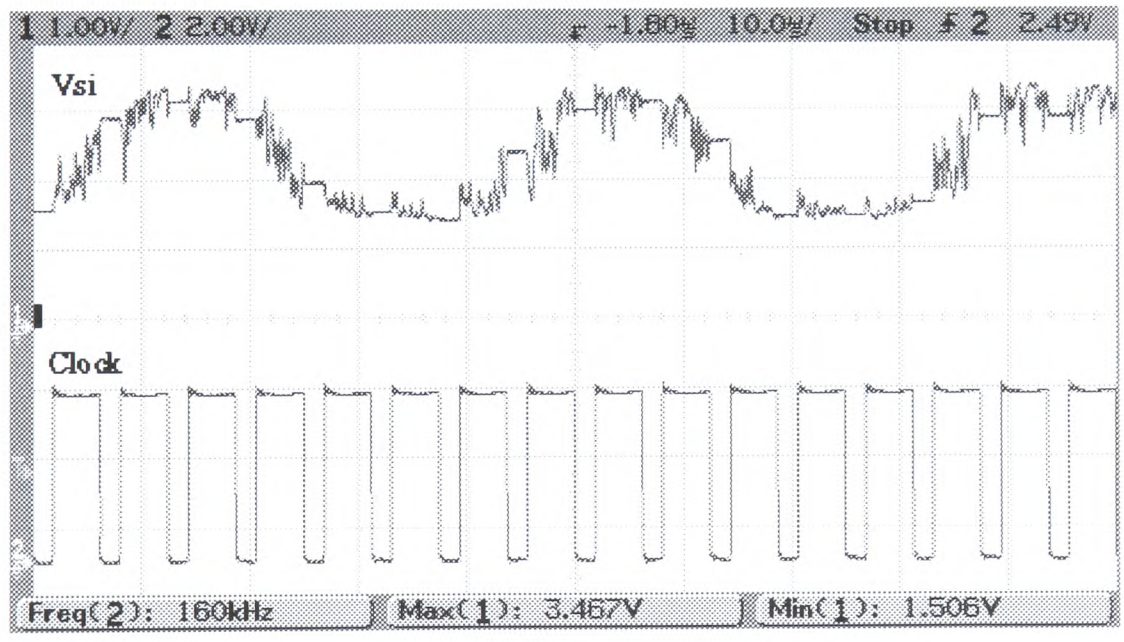
Measurement result

A CRBM neuron with five sets of inputs (i.e. $j = 5$ in Fig.6.9) was implemented in VLSI. The VLSI neuron's transient response, especially its probabilistic behaviour, is of this research's great interest. Fig.6.10 shows the measurement setup. The supply voltage is 5V and the reference zero for a voltage was set to 2.5V. Inputs $\{s_j\}$ were connected to a triangular wave sweeping between 1.5V and 3.5V (at 25kHz), while weights $\{w_{ij}\}$ were set to a constant voltage of 3.5V. A digital clock then sampled the neuron's output at 160kHZ with a duty cycle of 70%.

In the absence of a noise input and with $V_{ai} = 1.8V$, the measured output of a CRBM neuron is shown in Fig.6.11(a). Fig.6.11(a) reveals that $\{s_j\}$ and $\{w_{ij}\}$ drive the neuron's output to



(a)



(b)

Figure 6.11: *The measured output of a CRBM neuron in VLSI (a)in the absence of a noise input (b)with a noise input and* $V_\sigma = 3.8V$

sweep a sigmoid-shaped curve as shown in Fig.6.6. An output value, V_{si}, is sampled and hold whenever the switch is opened (Clock goes low). In contrast, Fig.6.11(b) shows the measured output of a CRBM neuron when a noise input is included, with $V_\sigma = 3.8V$ and $V_{ai} = 1.8V$. While $\{s_j\}$ and $\{w_{ij}\}$ still force the neuron's output to sweep a sigmoid-shaped curve, the noise input alters the curve dramatically. The neuron's output thus becomes continuous-valued probabilistic, and a continuous-valued V_{si} is sampled and held whenever the switch is opened. This pleasing result not only shows that a CRBM neuron is realised successfully in VLSI, but also supports the suggestion that "*continuous probabilistic behaviour can be realised effectively in VLSI circuits*". The approach is demonstrated to be the incorporation of artificially-generated noise. This noise-induced, continuous-valued probabilistic behaviour in VLSI is exactly what this thesis attempts to explore. A CRBM system composed of such VLSI neurons is expected to further adapt this noise-induced probabilistic behaviour.

In addition to measuring the stochastic behaviour of CRBM neurons in VLSI, a mapping between V_σ and the value of σ in simulation is derived. With the same noise input but various levels of V_σ, the peak-to-peak values of a neuron's output were measured when all $\{s_j\} = 2.5V$ (i.e. no deterministic input). Let $\triangle_n(V_\sigma)$ denote a measured peak-to-peak value for a particular V_σ. As 99.7% of samples from a Gaussian distribution lie in three standard deviations, the corresponding value of σ is estimated according to

$$\sigma(V_\sigma) = \frac{\phi^{-1}\left(\triangle_n(V_\sigma)/2\right)}{3} \tag{6.12}$$

where $\phi^{-1}(\cdot)$ is the inverse of the sigmoid function $\phi(\cdot)$. The derived mapping between V_σ and σ is shown in Fig.6.12. The mapping relationship is approximately linear, and this measurement result will be useful for setting up a CRBM system.

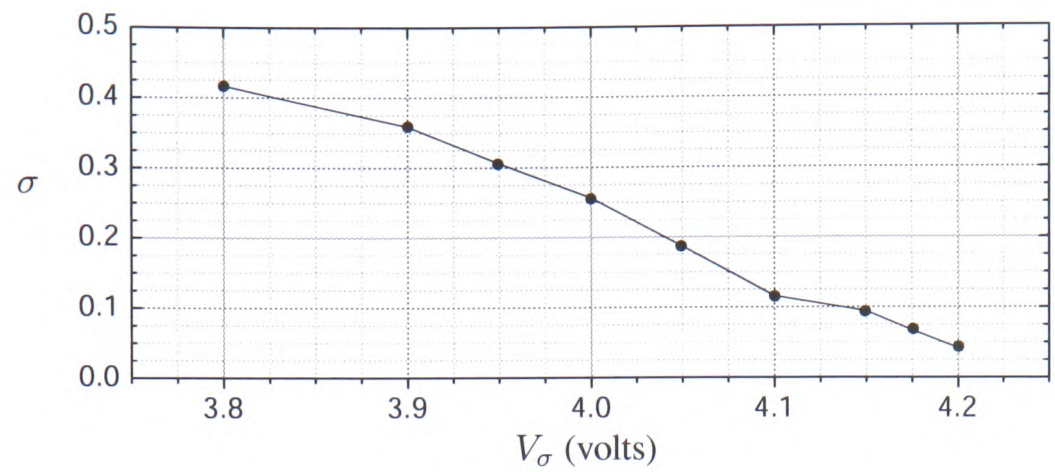


Figure 6.12: *The mapping for σ between hardware implementation and software simulation.*

6.5 Summary

Improved circuits for both a multiplier and a sigmoid-function circuit have been designed and implemented to facilitate continuous-valued probabilistic computation in VLSI. Measurement results indicate that these improved circuits fit a CRBM system's requirements well, i.e. suggesting solutions for the grey blocks in Fig.2.29. A noise generator is also implemented to provide 10 channels of near-Gaussian noise sources. Combining all these component circuits, a CRBM neuron has been realised in VLSI successfully. The detailed mappings for parameters $\{a_i\}$ and σ are also derived to facilitate the testing of a CRBM system in the future. Most encouragingly, a CRBM neuron in VLSI exhibits the continuous-valued probabilistic behaviour this research aims to explore, and has demonstrated the suggestion that continuous-valued probabilistic behaviour can be realised effectively in VLSI. It is expected that a CRBM system will further adapt this probabilistic behaviour in VLSI.

Chapter 7
Minimising-contrastive-divergence training in VLSI

This chapter concerns the VLSI implementation of the simplified MCD training rules for the CRBM , given in Eq.(5.1) and Eq.(5.2). As discussed in Sec.5.3, these simplified rules can be implemented according to the block diagram in Fig.2.30. The works in [52, 53] have suggested VLSI circuits for all blocks, most of which can be used directly to implement the MCD training rules for the CRBM. The only circuit to be improved is the weight-changing circuit, as highlighted by grey colours in Fig.2.30, while the use of the accumulator is already validated by the simulation in Ch.5. Therefore, after describing the VLSI circuits of basic arithmetic functions such as a subtractor briefly, this chapter discusses the design of a learning circuit and a voltage-limiter suitable for adapting the parameters of a CRBM system. The digital-control signals for training a CRBM system (Fig.5.6) are also implemented in VLSI (see Appendix A). Finally, combining all these component circuits yields two MCD training circuits, one for adapting $\{w_{ij}\}$ and the other for adapting $\{a_i\}$. The measurement results of on-chip MCD training will be presented and discussed.

7.1 Basic arithmetic functions

The basic arithmetic functions of a MCD training rule include a multiplier, a subtractor, and a sign function, as indicated by Eq.(5.1) and Eq.(5.2). VLSI circuits performing these functions have been suggested in [52,53]. However, the suggested multiplier is a two-quadrant multiplier, as the MCD training circuit in [53] was designed to train the PoE/RBM which has non-negative

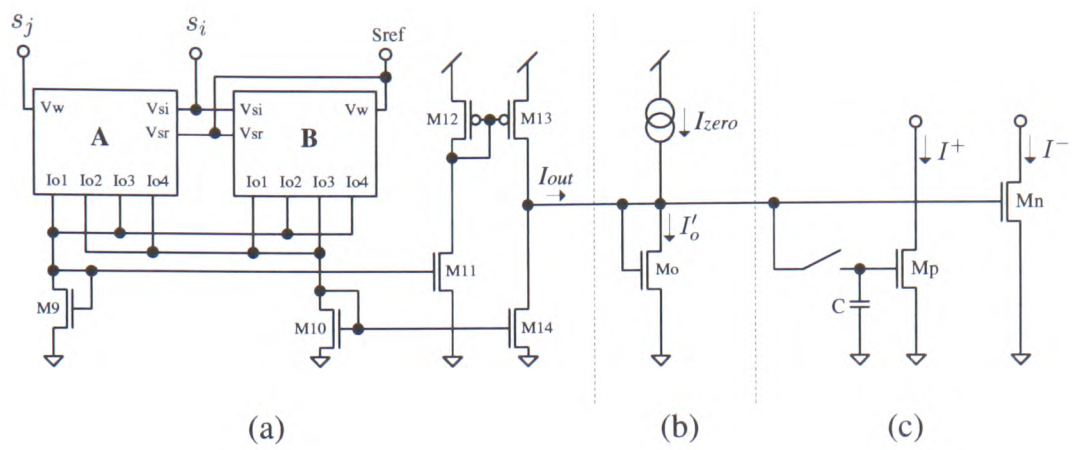


Figure 7.1: *The circuit that integrates (a)a modified Chible multiplier with (b)an output stage and (c)a dynamic current mirror. Component circuits are separated by dash lines*

$\{s_i\}$. The modified Chbile multiplier designed in Sec.6.1 is therefore employed to calculate signed $(s_i \cdot s_j)$ for a CRBM system, while adopting the subtractor and the sign circuit suggested in [52, 53]. All these circuits are described as follows.

Multiplier

Fig.7.1 shows a modified Chible multiplier together with an output stage and a dynamic current mirror. The multiplier outputs a current, I_{out}, proportional to $(s_i \cdot s_j)$. As shown in the measurement result in Fig.6.2, I_{out} is signed and thus bi-directional [1]. To ease computation, the output stage produces a current $I'_o = I_{out} + I_{zero}$, where I_{zero} is set to be the maximum absolute value of I_{out}. I'_o is thus non-negative and uni-directional with a reference zero at I_{zero}. Moreover, I'_o represents $(s_i \cdot s_j)$. The dynamic current mirror, Mp and C, then functions as a current memory, memorising a value of I'_o when a training datum is presented to a CRBM system. The drain current of Mp, I^+, thus represents the $(s_i \cdot s_j)$ calculated with the initial states of the CRBM system. As I'_o is also mirrored into Mn, the drain current of Mn, I^-, represents the $(s_i \cdot s_j)$ calculated after one step of Gibbs sampling. Both I^+ and I^- are subsequently fed into a current subtractor that calculates the difference between them, i.e. the contrastive divergence.

[1] Current components denoted in a circuit diagram are direction-wise. If a current is positive, the current flows in the direction as its arrow points towards, while if a current is negative, the current flows in the direction opposite to the arrow

Subtractor

As each contrastive divergence suggests an update direction for a parameter. it is important for a current subtractor to subtract I^- from I^+ as accurately as possible. Fig.7.2(a) shows the current subtractor suggested in [53]. The subtractor uses improved Wilson current mirrors [84] to mirror currents accurately. Transistors M1-M4, for example, constitute a Wilson current mirror. The sum of I_{bias} and I^+ is mirrored into M3 and M4, while I^- is mirrored into M5 and M6. As a result, the current I_d equals the difference between $(I^+ + I_{bias})$ and I^-, and is mirrored to the output transistors M15 and M16. The subtractor thus outputs a current of

$$I_{sub} = (I^+ - I^-) + I_{bias} \tag{7.1}$$

Eq.(7.1) indicates that the incorporation of I_{bias} makes I_{sub} non-negative and thus uni-directional. As both I^+ and I^- have reference zeros at I_{zero}, the most negative value of $(I^+ - I^-)$ is $-2I_{zero}$. I_{bias} is therefore set to be $2I_{zero}$. The output current of a current subtractor is fed into an current accumulator, as shown in Fig.2.27 and described in Sec.2.9. The accumulator subsequently calculates the average of four I_{sub} and outputs a current $I_{ave} = (\sum I_{sub})/4$ to a sign circuit.

Sign circuit

Fig.7.2(b) shows the sign circuit suggested in [53]. The sign circuit is simply a current comparator that compares I_{ave} to a reference current, I_{ref}, to determine an update direction, V_{comp}. As $I_{ave} = I_{sub}/4$ has its reference zero at I_{bias}, I_{ref} is set to have the same value as I_{bias}. The sign circuit mirrors I_{ave} and I_{ref} into transistors M6 and M4, respectively, producing a current $i_x = (I_{ave} - I_{ref})$. If $I_{ave} > I_{ref}$, i_x is positive, charging the gate voltages of both M7 and M8 towards V_{dd}. As M7 and M8 constitute an inverter, an input voltage close to V_{dd} causes $V_{comp} = 0V$. On the contrary, $I_{ave} < I_{ref}$ results in $V_{comp} = V_{dd}$. $V_{comp} = 0V$ therefore indicates an increase to a parameter, whereas $V_{comp} = V_{dd}$ indicates a decrease to a parameter.

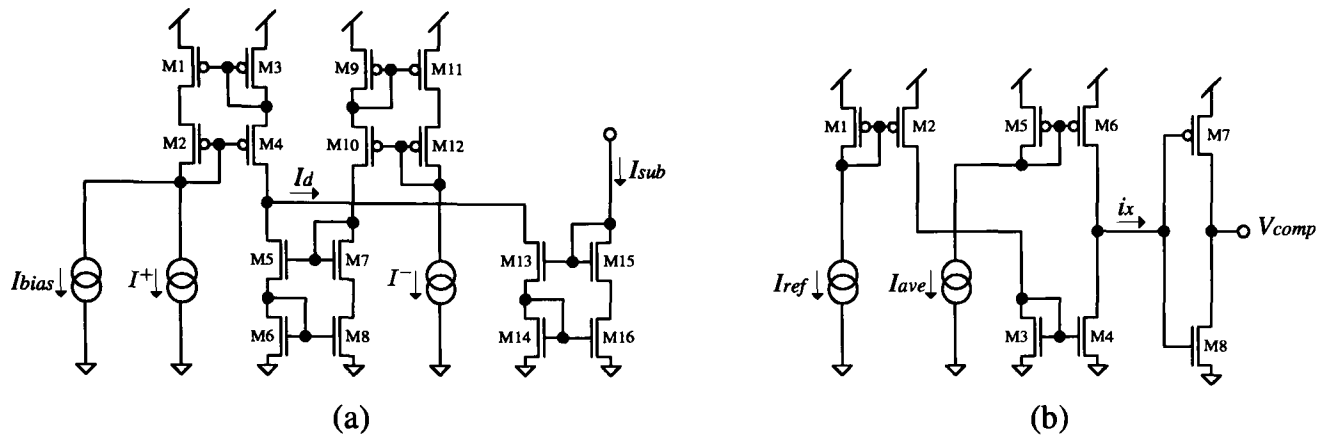


Figure 7.2: *The circuit diagrams of (a)the current subtractor (b)the sign circuit suggested in [53]*

This directional signal is sent to the learning circuit described in the following section.

7.2 Learning circuit

The weight-changing circuit proposed in [51,52] introduces a limited voltage range for parameters, as discussed in Sec.2.9. To provide parameters with the full range of a power supply, a pulse-controlled learning circuit, as shown in Fig.7.3, has been designed to adapt all parameters of a CRBM system. The pulse-controlled learning circuit is based on the learning cell proposed in [29] (Fig.7.3(a)), and uses a pulse generator (Fig.7.3(b)) to control the size of each learning step.

Learning cell

The learning cell in Fig.7.3(a) stores a parameter as a voltage on the capacitor C, and charges or discharges the capacitor according to $\overline{\text{INC}}$/DEC when the learning cell is enabled ($EN = 1$). Let logic-1 correspond to V_{dd}, and logic-0 to 0V. If $\overline{\text{INC}}$/DEC $= 0$, the source voltages of both Mp and Mn equal V_{dd}. Mp thus conducts a finite current to charge the capacitor while Mn is switched off. Inversely if $\overline{\text{INC}}$/DEC $= 1$, the source voltages of both Mp and Mn equal 0V. Mn then conducts a non-zero current to discharge the capacitor while Mp is switched off. As

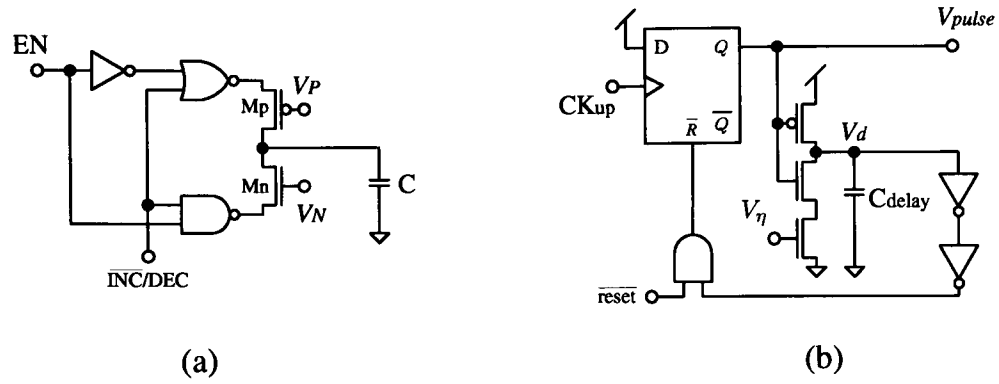


Figure 7.3: *The pulse-controlled learning circuit comprised of (a)a learning cell proposed in [29] and (b)a width-variable pulse generator.*

the capacitor can be charged or discharged until the drain-source voltage of Mp or Mn becomes zero, the capacitor has a voltage range of almost 0V to V_{dd}.

Compared to the weight-changing circuit in [51, 52], the learning cell not only allows parameters to have voltage ranges from 0V to V_{dd}, but also has a simpler architecture. Furthermore, the learning cell can function as a *dynamic analogue memory* to refresh a parameter to a constant analogue level [91, 92] [2]. With an A/D converter, the analogue voltage stored in the learning cell can be converted into a binary value, and compared with a reference binary value which represents a desired analogue level. The comparison then yields a refresh signal for instructing the learning cell to increase or decrease the analogue voltage towards the desired level. As comparison between binary values can be carried out with digital circuits, the dynamic analogue memory compares favourably with other analogue-memory schemes by avoiding the need for extra D/A converters or analogue resources [91, 92]. The learning cell therefore suggests a useful training circuit for neural systems, as demonstrated in [29].

In addition to the advantages above, the ability to adapt parameters with small steps is important to minimise the effect of training errors, as discussed in Sec.2.9. Fig.7.3(a) shows

[2] Analogue voltages stored on capacitors vary with time because of leakage currents to reverse-biased diodes that are parasitic at transistors' source or drain [84]. Therefore, capacitively-stored analogue memories normally require a refresh scheme.

how V_P and V_N control the gate voltages of Mp and Mn, and thus the charging and discharging current to the capacitor C. This feature renders update stepsizes adjustable through V_P and V_N [29]. However, transistor nonlinearities and process variations make it difficult to control the charging or discharging current precisely, discouraging the use of different stepsizes for various parameters, as a CRBM system requires. Nevertheless, the works in [52, 53] suggest the use of a pulse generator as an alternative way to control learning rates. The learning rates for a CRBM system are therefore designed to be controlled by applying a width-variable pulse to the input EN, which subsequently controls the charging or discharging *time* for the capacitor C. As a stepsize is linearly proportional to its charging time, the pulse-control scheme allows various stepsizes to be set precisely and easily by monitoring the pulse width off-chip. A pulse generator is shown in Fig.7.3(b) and described as follows.

Pulse generator

The pulse generator is largely a D-type flip-flop whose output V_{pulse} is initially reset low via $\overline{\text{reset}}$. V_{pulse} then goes high on the rising edge of CKup, while the capacitor C_{delay} prevents V_d from changing instantly. Eventually, V_{pulse} is reset to zero as soon as V_d is discharged and a positive pulse is generated. V_η controls the discharging current for C_{delay} and thus the width of the generated pulse. V_η therefore controls the learning rates η_w and η_a in Eq.(5.1) and Eq.(5.2). As the input capacitance of each learning cell is less than 0.1pF, one pulse generator can control all learning cells with the same learning rate (update stepsize). The simulation in Sec.5.2 implies that the CRBM system this research aims to implement requires only three different learning rates, η_w, η_{av}, and η_{ah}. Therefore, three pulse generators can control all learning cells of the CRBM system. In addition, the measurement results in Sec.7.4 will reveal that the pulse width can vary from 6.8ns to more than 5s, offering a good range of learning rate.

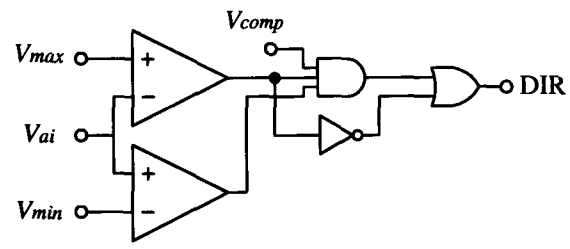


Figure 7.4: *The voltage-limiting circuit*

7.3 Voltage-limiting circuit

Although the learning circuit allows parameters to have voltages from 0V to V_{dd}, the voltages of parameter $\{a_i\}$ (i.e. V_{ai}) must be confined to $[1,3]$V, to ensure the normal operation of the voltage-controlled feedback resistors. A voltage-limiting circuit as shown in Fig.7.4 is thus designed to limit the range of V_{ai}, with V_{max} and V_{min} defining the range through two voltage comparators. V_{comp} is the output from a sign circuit. If $V_{max} > V_{ai} > V_{min}$, DIR equals V_{comp}, i.e. the MCD training rule decides the update direction. However, if $V_{ai} > V_{max} > V_{min}$, DIR goes high to force a decrease in V_{ai}, while if $V_{max} > V_{min} > V_{ai}$, DIR goes low to force an increase in V_{ai}.

7.4 On-chip MCD training

Two MCD training circuits, one for $\{w_{ij}\}$ the other for $\{a_i\}$, have been implemented by combining all the component circuits introduced in previous sections. Fig.7.5 shows the block diagram of the MCD training circuits. The training circuit for $\{a_i\}$ differs from that for $\{w_{ij}\}$ by the inclusion of a voltage-limiting circuit, as indicated by the dash-lined box in Fig.7.5. In addition, s_j and $\hat{s_j}$ in Fig.7.5 are replaced by s_i and $\hat{s_i}$, respectively, for training $\{a_i\}$. In learning mode (LER/$\overline{\text{REF}}$ = 1), the initial states s_i and s_j are first sampled by clock signals CKdata and CKH0, resulting in a current I^+ at the output of the multiplier. After CK+ samples and holds I^+, the one-step reconstructed states $\hat{s_i}$ and $\hat{s_j}$ are sampled by CKV and CKH1 to

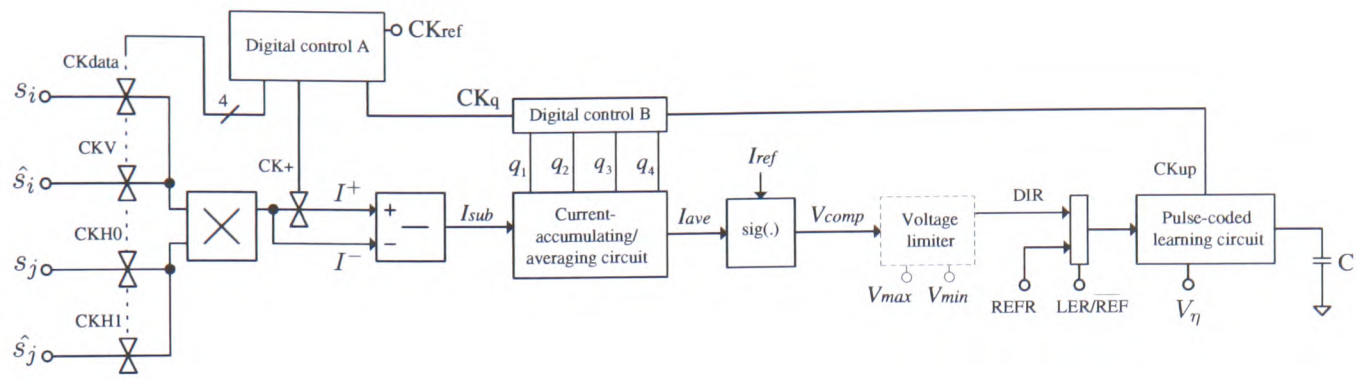
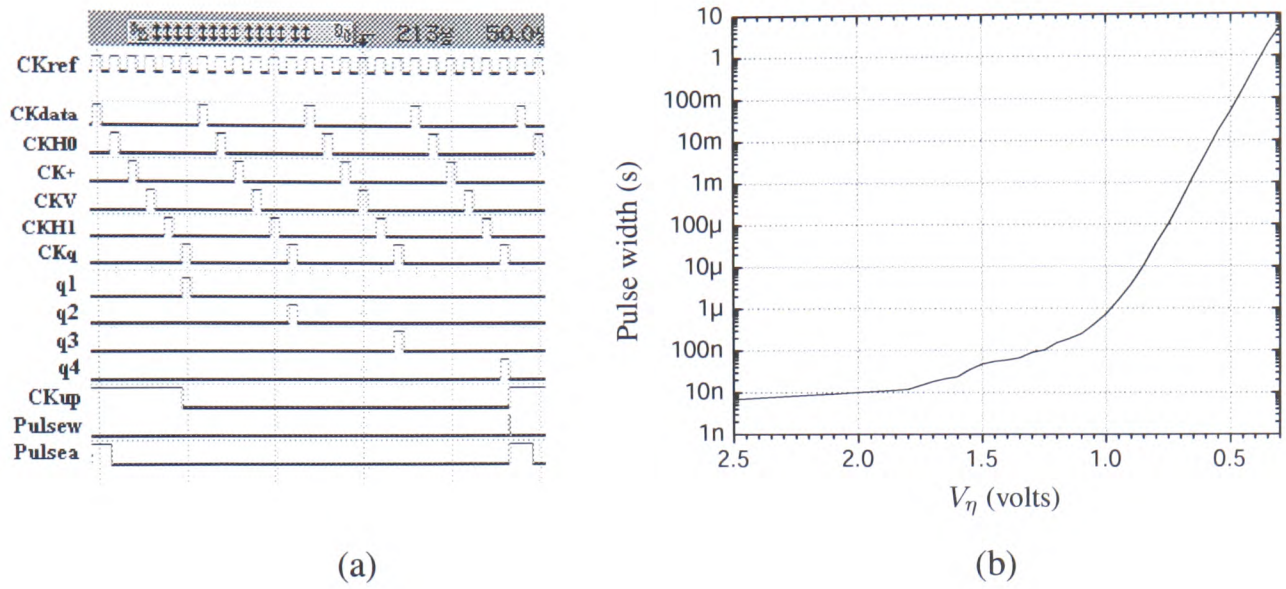


Figure 7.5: *The block diagram of the MCD training circuit for training a CRBM system*

produce another current I^-. CKq then triggers the accumulator to sample and hold the output of the current subtractor, I_{sub}. By repeating the above clocking sequence for *four* cycles, four I_{sub} are accumulated and averaged to derive I_{ave}, representing $\langle s_i s_j \rangle_4 - \langle \hat{s}_i \hat{s}_j \rangle_4$ in Eq.(5.1) or $\langle s_i^2 \rangle_4 - \langle \hat{s}_i{}^2 \rangle_4$ in Eq.(5.2). Finally, the sign circuit compares I_{ave} to I_{ref} to determine the update direction DIR, and the learning circuit, triggered by CK_{up}, updates the parameter once. In refreshing mode (LER/REF $= 0$), the signal REFR rather than DIR determines the update direction, maintaining the parameter to a reference value. The measurement results of the MCD training circuits are presented and discussed as follows

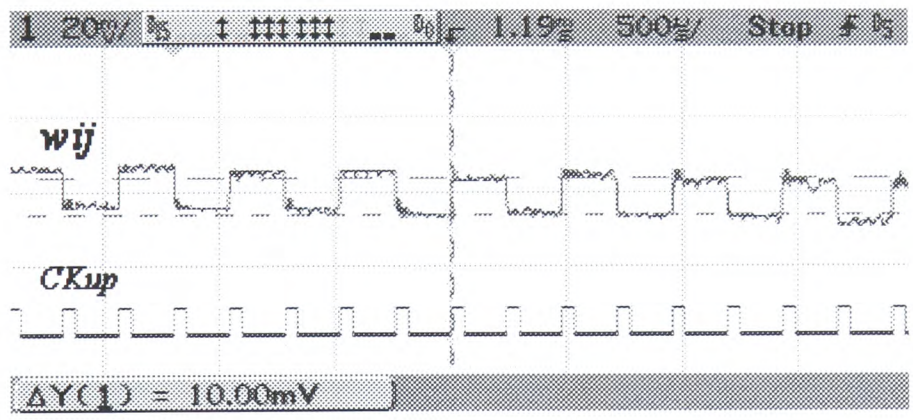
Fig.7.6(a) shows the measured digital signals from a fabricated chip. The first six clocks, from CKdata to CKq, activate sequentially in agreement with Fig.5.6(a). One of q1-q4 then activates individually whenever the six clocks run through their sequence once. As soon as q4 completes its activation, CKup turns on and triggers two pulse generators to produce two pulses with different widths. The measured relationship between a pulse width and its controlling voltage V_η is displayed in Fig.7.6(b). As V_η changes from 2.5V to 0.3V, the pulse width varies from as short as 6.8ns to as wide as 5.28s, corresponding to a range of several decades for learning rates (update stepsizes).

Fig.7.7 shows the measured update stepsizes of a learning circuit in refreshing mode. With

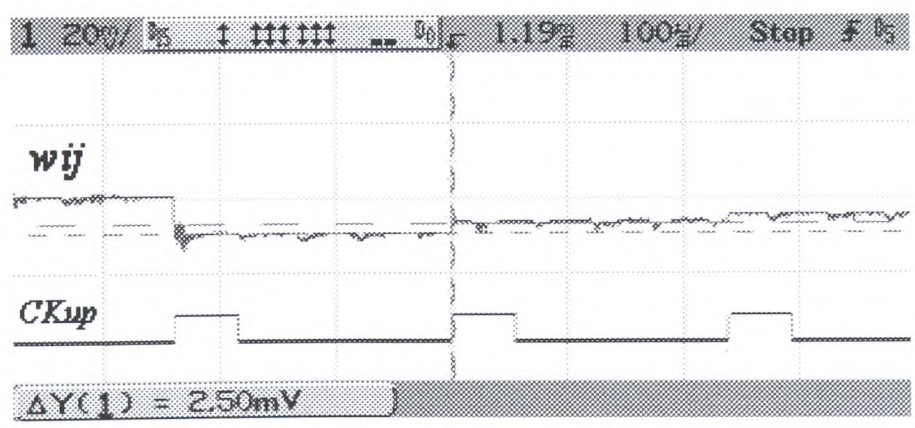


(a) (b)

Figure 7.6: *(a)The measured digital-control signals for training a CRBM system (b)The measured relationship between a pulse width and its controlling voltage V_η.*



(a)



(b)

Figure 7.7: *(a)A measured update stepsize of 10mV with $V_P = 4.07V$, $V_N = 0.7V$, and a pulse width of 300ns (b)A measured update stepsize of 2.5mV with a pulse width of 75ns.*

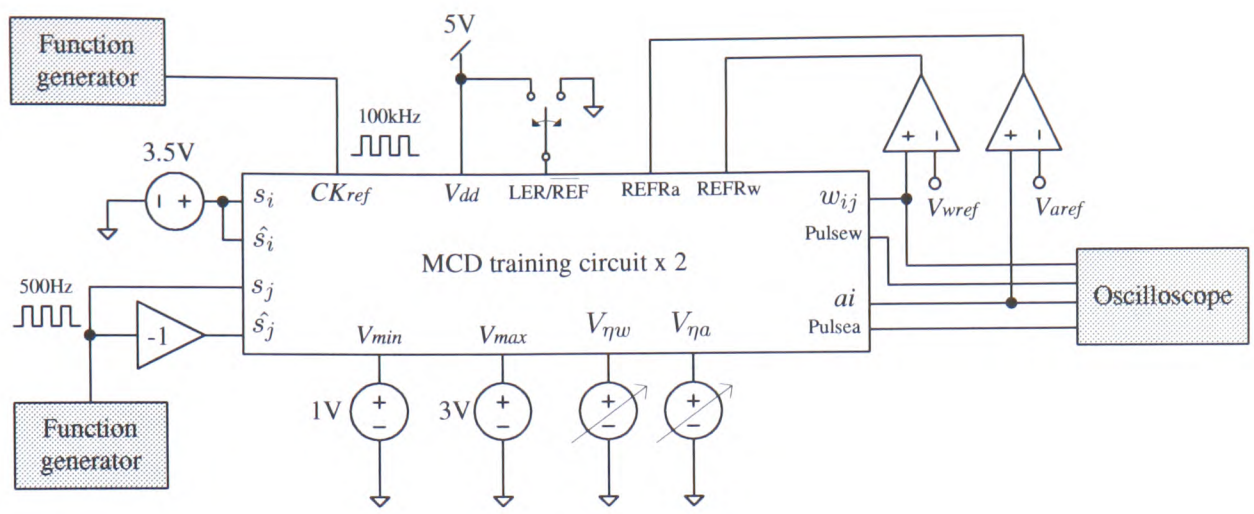


Figure 7.8: *The measurement setup for testing full MCD training circuits for both* $\{w_{ij}\}$ *and* $\{a_i\}$.

$V_P = 4.07V$, $V_N = 0.7V$, and a pulse width of 300ns, the measured stepsize is 10mV in both increasing and decreasing direction, as shown in Fig.7.7(a). Using this measurement as a reference, various stepsizes are easily obtainable by adjusting the pulse width according to Fig.7.6(b). For example, with a pulse width of 75ns and the same voltage for V_P, an increasing stepsize as small as 2.5mV is achieved easily, as depicted in Fig.7.7(b). A stepsize smaller than 2.5mV is also achievable, whereas noises coupled on a capacitor ($\sim$2mV) make such a tiny stepsize hardly observable in measurements.

Fig.7.8 shows the measurement setup for testing the MCD training circuits for both $\{w_{ij}\}$ and $\{a_i\}$. The power supply voltage is 5V, and the reference zero for a voltage is 2.5V. To ease testing, s_i and $\hat{s_i}$ are fixed at 3.5V(corresponding to $s_i = 1$ in simulation), while a square wave alternating between 3.5V and 1.5V(corresponding to $s_i = -1$ in simulation) is sent to s_j and $\hat{s_j}$. An inverting, unit-gain amplifier introduces complementary inputs to s_j and $\hat{s_j}$, i.e. $s_j = -\hat{s_j}$. With such inputs, parameters should learn downwards when $s_j = 3.5V$ and $\hat{s_j} = 1.5$, and upwards when $s_j = 1.5V$ and $\hat{s_j} = 3.5$, as indicated by the training rule in Eq.(5.1). In refresh mode, the voltage comparators generate refresh signals for the learning circuits to maintain parameters at particular levels (V_{wref} and V_{aref}). The digital-control circuits are clocked at a

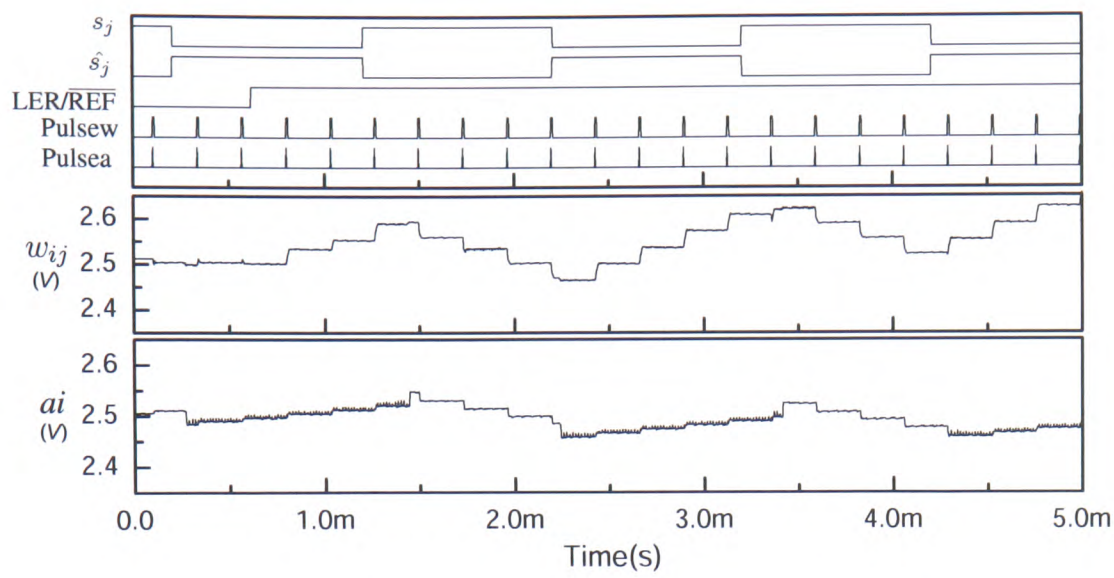


Figure 7.9: *Measured on-chip training of both w_{ij} and a_i with different learning rates. The update stepsize is 34mV for w_{ij}, and 10mV for a_i.*

reference frequency of 100kHz.

Fig.7.9 shows the measured on-chip training of both w_{ij} and a_i with different learning rates. Both parameters were initially refreshed to 2.5V when LER/$\overline{\text{REF}} = 0$, and subsequently started to learn up and down in response to the changing s_j and $\hat{s_j}$ when LER/$\overline{\text{REF}} = 1$. As controlled by different pulse widths, the two parameters were updated with different stepsizes (34mV and 10mV)[3], albeit in the same direction. The trace of parameter a_i exhibits digital noise attributable to sub-optimal layout and has been improved in a subsequent design.

Fig.7.10 shows the measured on-chip training of both w_{ij} and a_i in different directions. Both parameters were refreshed to 3.5V and the maximum voltage for a_i was set to 3V. Therefore, when LER/$\overline{\text{REF}} = 1$, the voltage-limiting circuit forced a_i to decrease towards V_{max}, whereas w_{ij} continued to train up and down. The promising measurement results in Fig.7.9 and Fig.7.10 indicate that the MCD training circuits for a CRBM system have been implemented successfully.

[3]With V_P=4.07V and V_N=0.7V, the pulse width for w_{ij} was 1us, and that for a_i was 300ns.

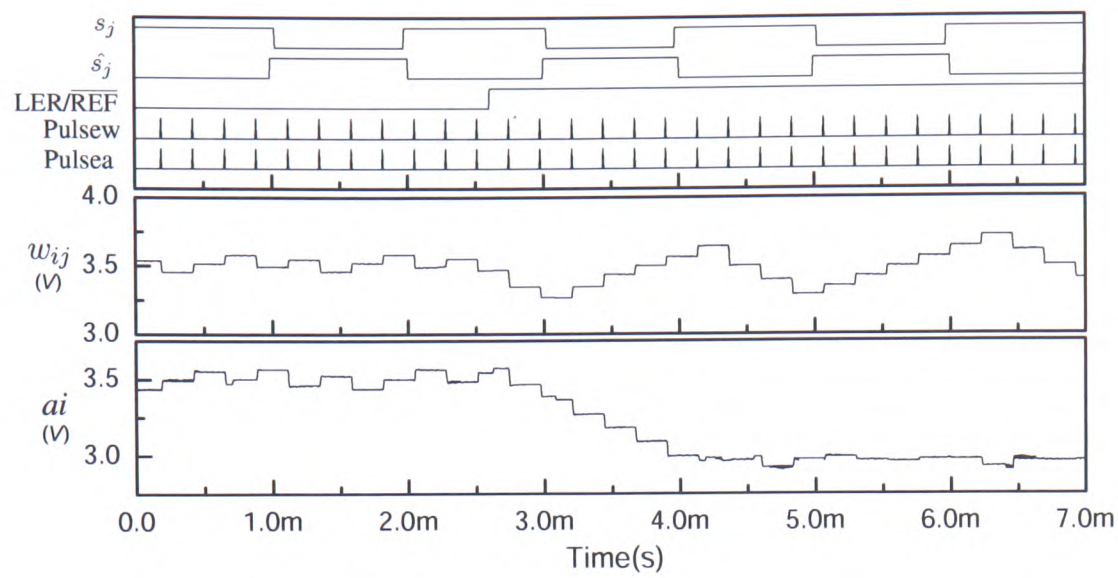


Figure 7.10: *Measured on-chip training of both w_{ij} and a_i with different training directions. The update stepsizes for both parameter are 60mV.*

7.5 Summary

Components of MCD training circuits, from basic arithmetic functions to digital-control signals, for a CRBM system have been implemented in VLSI. Among these component circuits, the notable contribution is the design of a pulse-controlled learning circuit, which not only provides parameters with the full range of a power supply but also eases the setting of learning rates for parameters. The pulse-controlled learning circuit is therefore suitable for adapting the parameters of a CRBM system, i.e. providing a solution for filling the grey block (of the weight-changing circuit) in Fig.2.30. Combining all component circuits, the MCD training circuits for both $\{w_{ij}\}$ and $\{a_i\}$ are realised in VLSI. The promising measurement results reveal that the MCD training circuits are able to adapt parameters with variable stepsizes and in a correct direction. As both the stochastic neurons and the MCD training rules for the CRBM have been implemented successfully, this research proceeds to implement a full CRBM system, in order to study the utility and on-chip adaptability of continuous-valued probabilistic behaviour in VLSI.

Chapter 8
A Continuous Restricted Boltzmann Machine in VLSI

A full CRBM system with on-chip adaptability is implemented in VLSI, integrating the VLSI neurons and MCD training circuits designed in Ch.6 and Ch.7. This chapter first describes the architecture of a full CRBM system, and then the system's ability both to model and to regenerate continuous data distributions. In a preliminary training test, a CRBM system was trained to model continuous data sampled from a single-Gaussian distribution. This preliminary test, however, yielded non-ideal training results. The cause of the non-ideal training will be investigated and discussed. With non-ideal training circuits, a CRBM system's ability to model diverse continuous data will also be studied. Finally, a CRBM system's performance in regenerating continuous data distributions, through its noise-induced probabilistic behaviour, will be explored.

8.1 A full system on chip

The CRBM system implemented in VLSI contains two visible and four hidden neurons, with 20 MCD training circuits for adapting the parameters of all neurons. The modular diagram in Fig.9.1 shows the architecture of the system. Though largely similar to the modular diagram in Ch.5, Fig.9.1 further includes the control signal LER/$\overline{\text{REF}}$ and the external voltage comparators. This highlights the fact that all parameter values can not only be traced during training (LER/$\overline{\text{REF}} = 1$), but also be refreshed to particular analogue levels in *recall* mode [1]

[1]The recall mode for a CRBM system refers to Gibbs sampling from the equilibrium distribution modelled by the system

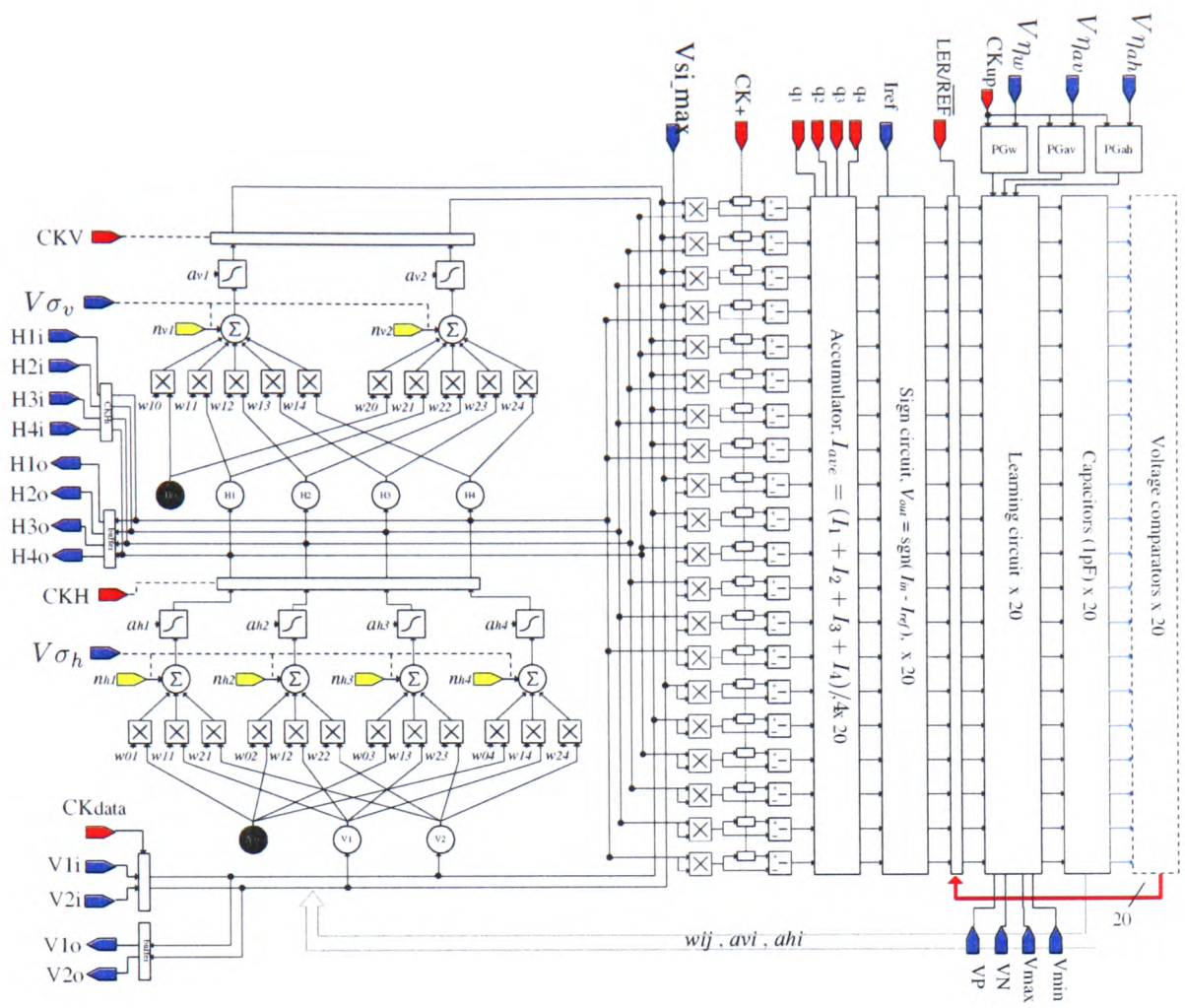


Figure 8.1: *The architecture of a CRBM system on chip.*

(LER/$\overline{\text{REF}}$ = 0). In addition, the inputs H1i-H4i allow the states of hidden neurons to be set externally, facilitating the testing of each neuron or each MCD training circuit individually. The digital-control circuits for training the system are implemented on the same chip. To enhance testing flexibility, all digital-control signals can be monitored off-chip or be replaced by external digital signals. Finally, the noise inputs of neurons are designed to be provided by the noise generator in Ch.6.

Fig.8.2 shows the chip layout of a full CRBM system. The power consumption and layout areas of both the full system and individual circuits are summarised in table.8.1. Although the full system consumes a total current of 19.644mA, most current consumption is attributable to the analogue buffers that read out parameter values, as shown in Fig.8.2. Excluding the 20

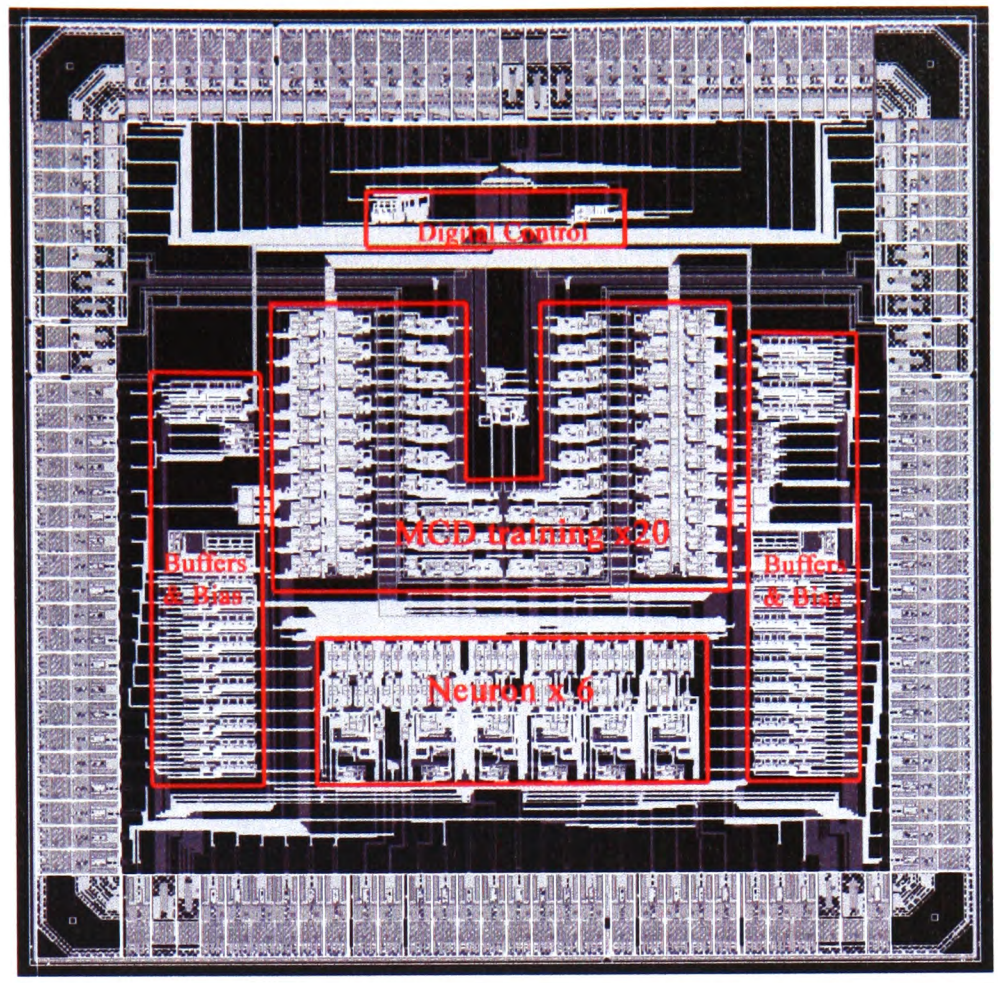


Figure 8.2: *The chip layout of a full CRBM system.*

	Power supply current (mA)	Circuit Area ($\mu m \times \mu m$)
Visible neuron	0.941	367×523
Hidden neuron	0.876	257×523
MCD training for $\{w_{ij}\}$	0.109	1128×145
MCD training for $\{a_i\}$	0.255	898×145
Analogue buffer	0.624	327×87
Full CRBM system	19.644	3197×2928

*Power supply currents are measured at $V_{dd} = 5V$

Table 8.1: *The power consumption and area of both a full CRBM system and its component circuits. The current consumption of the full system is measured from fabricated chips, while that of individual circuits is estimated from simulation results.*

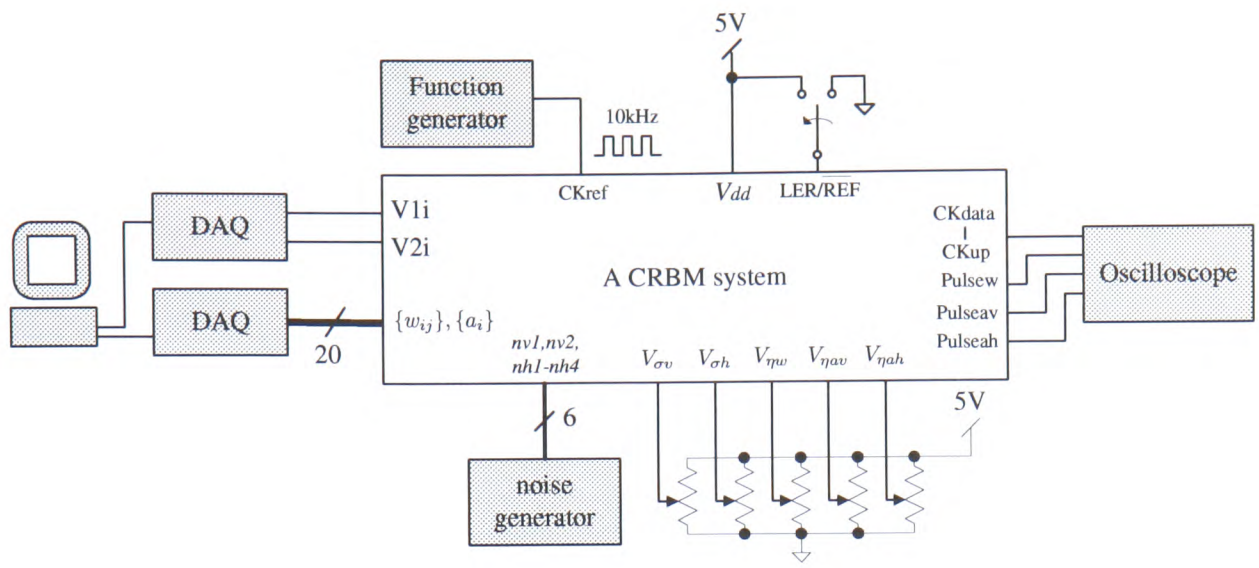


Figure 8.3: *The measurement setup for training a CRBM system.*

analogue buffers, the system consumes a current of only 7.164mA.

Fig.8.3 shows the measurement setup for training a CRBM system. The PC programs two data-acquisition (DAQ) boards to send initial data to visible neurons and to trace the values of all parameters. As a DAQ board can only update its analogue outputs at a rate lower than 10kHz, the function generator clocks the CRBM system at a reference frequency of 10kHz. The oscilloscope then monitors all digital signals, especially the digital pulses that control the learning rates of parameters. In addition, the array of potentiometers produces analogue reference voltages for defining the learning rates ($V_{\eta w}$, $V_{\eta av}$, $V_{\eta ah}$) and the noise variances ($V_{\sigma v}$, $V_{\sigma h}$) of the system. The mapping between analogue voltages and these constants is given in Fig.6.12 and Fig.7.6(b). The following sections present and discuss the measurement results of training a CRBM system to model continuous data distributions.

8.2 Modelling a single cluster of data points

A CRBM system was first trained with the two-dimensional data shown in Fig.8.4, containing 20 data points sampled from a single, circular Gaussian distribution. The training stepsizes for

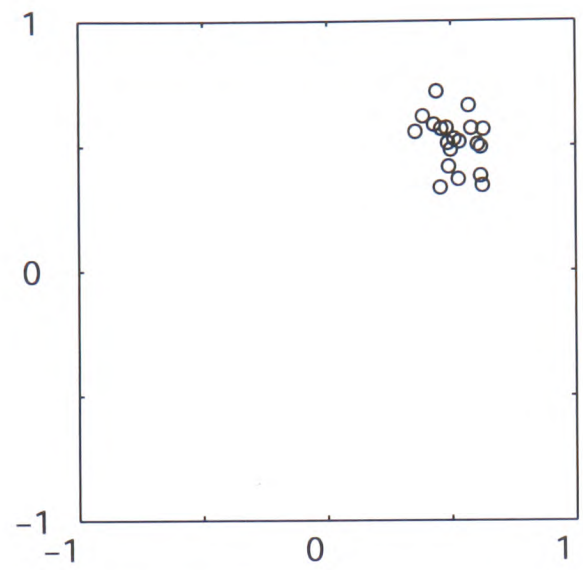


Figure 8.4: *20 training data points sampled from a single Gaussian distribution.*

$\{w_{ij}\}$ and the $\{a_i\}$ for hidden neurons were initially set as 3mV, while that for the $\{a_i\}$ for visible neurons was set to a higher value of 15mV, as suggested by the simulation in Ch.5. Both $V_{\sigma v}$ and $V_{\sigma h}$ were set to 4.05V, corresponding to $\sigma = 0.2$ in simulation (Fig.6.12).

A pre-test however revealed two non-ideal results. The first was that sending noise from another chip (on another PC board) coupled non-negligible noise ($\sim$150mV) onto all analogue reference voltages. This prevents the pulse width of a pulse generator from being controlled accurately via V_η. To preserve accurate control over learning rates, function generators were used as noise sources. The second non-ideal result was that most $\{w_{ij}\}$ tended towards their minimum limit (0V), while most $\{a_i\}$ tended towards their maximum limit (3V). In an attempt to compensate these tendencies, the increasing stepsize for $\{w_{ij}\}$ was adjusted to 15mV, and the decreasing stepsizes for $\{a_i\}$ were adjusted to 15mV and 45mV for hidden and visible neurons, respectively.

With the modified settings, a CRBM system was trained for 1000 epochs (updates) and the measured traces of parameters are shown in Fig.8.5. Most parameters still tended to reach their minimum or maximum limits, despite the introduction of "customised" training stepsizes. To

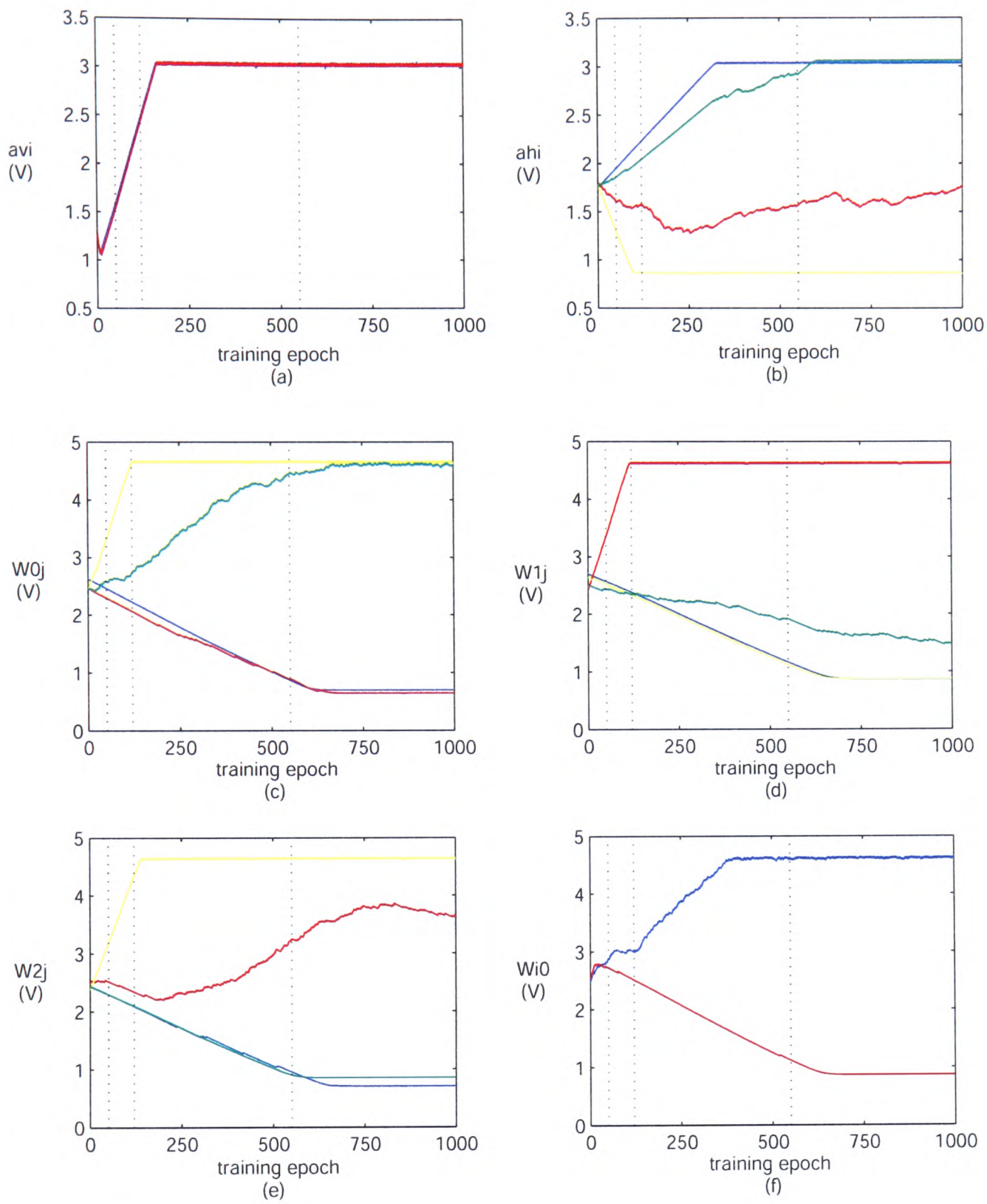


Figure 8.5: *The measured traces of* $\{a_i\}$ *and* $\{w_{ij}\}$ *when a CRBM system was trained with the data in Fig.8.4 for 1000 epochs (a)*$\{a_{vi}\}$ *(b)*$\{a_{hi}\}$ *(c)*$\{w_{0j}\}$ *(d)*$\{w_{1j}\}$ *(e)*$\{w_{2j}\}$ *(f)*$\{w_{i0}\}$*. In each subplot, blue, red, yellow, and green curves correspond to* i*(or* j*)= 1, 2, 3, and 4, respectively. The same colour conventions will be used in the following of this thesis. The grey dotted lines highlight the parameter values used to generate 20-step reconstructions in Fig.8.6.*

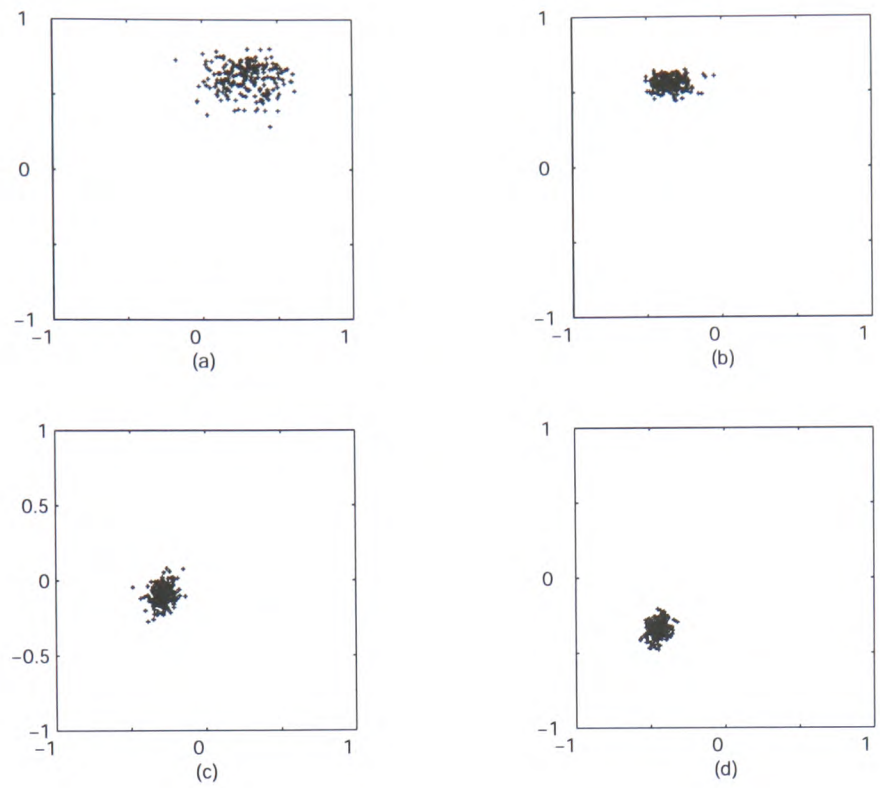


Figure 8.6: *20-step reconstructions generated by a CRBM model in Matlab with parameters values learnt by a CRBM system after (a)50 (b)120 (c)550 (d)1000 epochs*

investigate the distributions modelled by the CRBM system, the parameter values learnt after 50, 120, 550, and 1000 epochs were *substituted*[2] into a CRBM model in Matlab and generated the 20-step reconstructions in Fig.8.6. The reconstruction at epoch 50 (Fig.8.6(a)) approximates the training data roughly, indicating that the CRBM system did adapt its parameters towards minimising contrastive divergences between modelled distributions and the distribution of training data. However, as several parameters (w_{04}, w_{12}, w_{24}, a_{h4}, a_{v1}, and a_{v2}) reach their limits at epoch 120, the reconstructed cluster begins to "drift" away from its initial position (Fig.8.6(b)). At epoch 550, the reconstructed cluster further moves towards the bottom-left corner (Fig.8.6(c)), as a result of that more parameters reach limits. Finally, most parameters reach their limits after 1000 epochs. Fig.8.6(d) shows that the reconstruction at epoch 1000 remains a single, circular cluster of data points, whereas the position of the cluster differs significantly from that of the training cluster. This experiment result indicates that although the CRBM sys-

[2]Mapping $\{w_{ij}\}$ from hardware to software simply adopts a 1-to-1 ratio, as specified in table.5.1, while the mapping curve for $\{a_i\}$ is given in Fig.6.7.

tem did attempt to minimise contrastive divergences, non-ideal effects prevented the CRBM system from achieving a minimum divergence, i.e. from modelling the training data optimally. The source of the non-ideal effects are investigated and discussed in the next section.

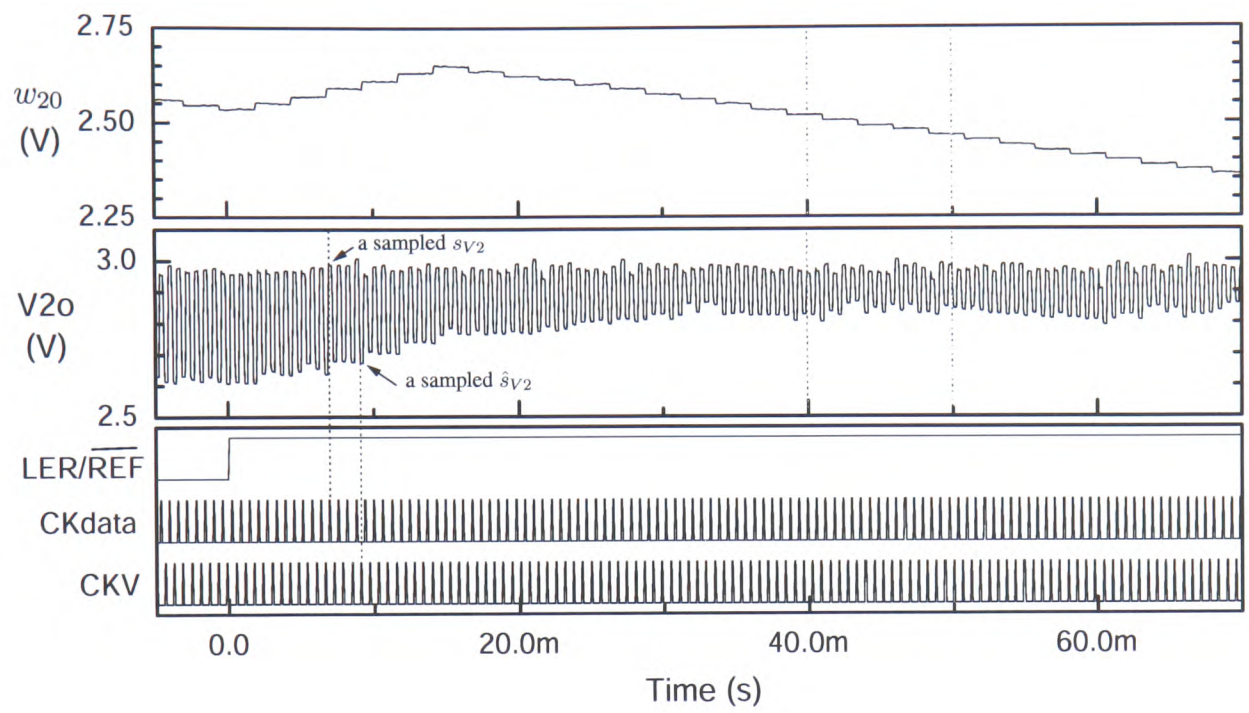
8.3 Non-ideal training in a CRBM system

8.3.1 The cause of non-ideal training

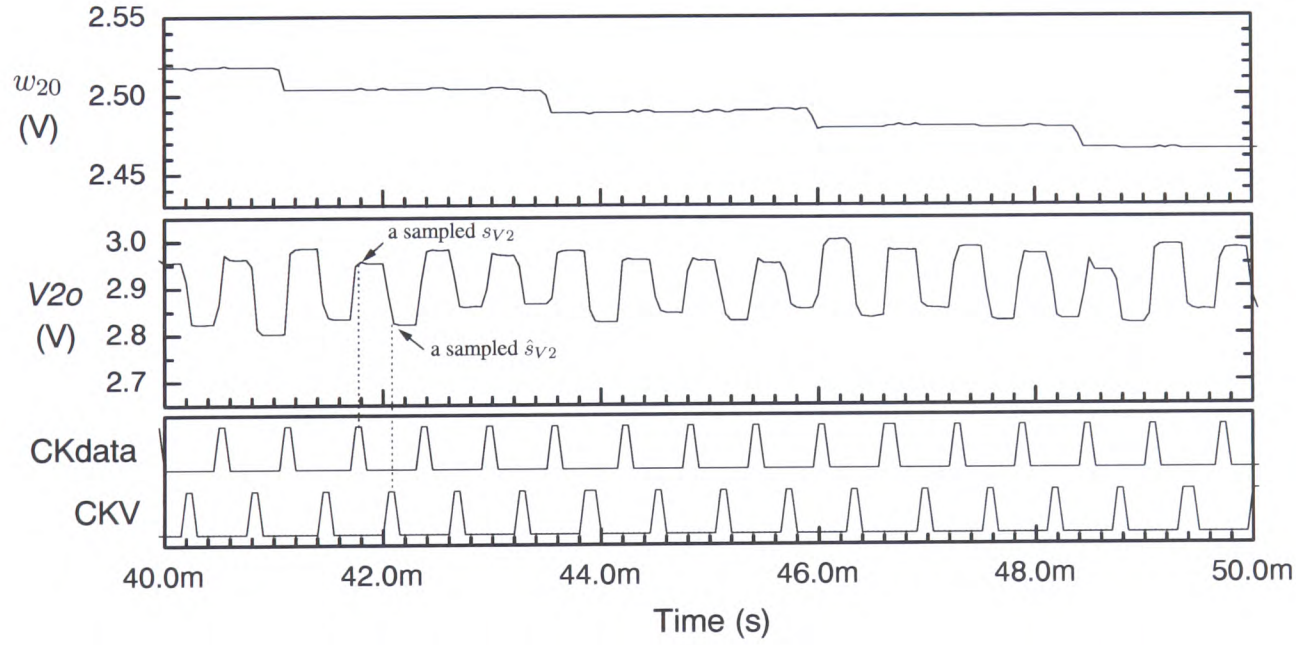
To investigate why most parameters tend to approach their maximum or minimum limits, the training task was simplified as presenting only one training datum, $(V1i, V2i) = (3, 3)(V)$, to the visible neurons. The trace of w_{20} and the output of visible neuron V2 (V2o) were then monitored during a training process. Ideally, the updating direction for w_{20} follows

$$\Delta w_{20} \propto sgn(\langle s_{V2} \rangle_4 - \langle \hat{s}_{V2} \rangle_4) \tag{8.1}$$

where s_{V2} and $\hat{s}_{V2}$ denote the initial and the one-step sampled outputs of V2, respectively, and $\langle \cdot \rangle_4$ denotes averaging over four data. As the single training datum sets $s_{V2} = 3V$ continuously, Eq.(8.1) indicates that w_{20} should always increments as long as $\hat{s}_{V2} < 3V$. Fig.8.7 shows the measured w_{20}, V2o, and relating digital-control signals. Fig.8.7(a) shows that CKdata and CKV activates alternately to sample V2o as initial data, s_{V2}, and as a one-step sample, $\hat{s}_{V2}$. $\hat{s}_{V2}$ is obviously smaller than s_{V2} before the training starts (LER/$\overline{\text{REF}}$ = 0). As soon as LER/$\overline{\text{REF}}$ goes high, w_{20} increments for several steps as predicted by Eq.(8.1), and the difference between s_{V2} and $\hat{s}_{V2}$ decreases gradually. However, w_{20} starts to decrement after 15ms and the difference between s_{V2} and $\hat{s}_{V2}$ subsequently stops decreasing. For a more careful inspection, the measured results between 40ms and 50ms are magnified and displayed in Fig.8.7(b). Fig.8.7(b) reveals clearly that s_{V2} remains larger than $\hat{s}_{V2}$ during this period, while w_{20} actually decrements. This leads to the suspicion that the training circuit can not



(a)



(b)

Figure 8.7: *(a)Measured w_{20} and $V2$ when only one training datum, $(V1i, V2i) = (3, 3)(V)$, is presented to visible neurons. (b)Zooming-in of the period between 40ms and 50ms in (a).*

determine a correct updating direction when the contrastive divergence, $s_{V2} - \hat{s}_{V2}$, is smaller than some threshold value (e.g. 200mV).

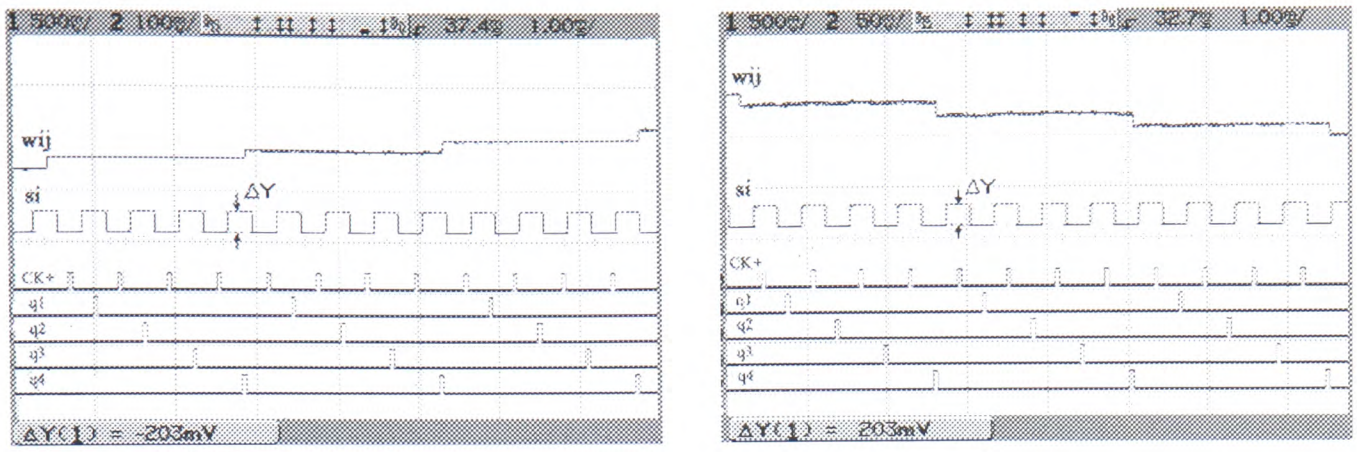
Let s_i and s_j denote the two inputs of a training circuit which calculates contrastive divergences as $(s_i \cdot s_j) - (\hat{s_i} \cdot \hat{s_j})$ [3]. To verify this suspicion, each training circuit was tested individually with $s_j = \hat{s_j} = 3.5V$ and s_i receiving a square wave which had an offset voltage of 2.5V and an amplitude Δ_a. In addition, the square wave had the same frequency as CK+ and CKq, which sample $(s_i \cdot s_j)$ and $(\hat{s_i} \cdot \hat{s_j})$, respectively. So if CK+ samples at $s_i = (2.5 + \Delta_a)V$ and $s_j = 3.5V$, CKq must sample at $\hat{s}_i = (2.5 - \Delta_a)V$ and $\hat{s_j} = 3.5V$, and vice versa. As $s_j = \hat{s_j} = 3.5V$ is equivalent to $s_j = \hat{s_j} = 1$ in Matlab simulation, each training circuit calculates an expected contrastive divergence as

$$\begin{aligned} \Delta_D &= \langle s_i \cdot s_j \rangle_4 - \langle \hat{s_i} \cdot \hat{s_j} \rangle_4 \\ &= \begin{cases} +2\Delta_a & \text{if} \quad s_i = 2.5 + \Delta_a, \hat{s_i} = 2.5 - \Delta_a \\ -2\Delta_a & \text{if} \quad s_i = 2.5 - \Delta_a, \hat{s_i} = 2.5 + \Delta_a \end{cases} \end{aligned} \tag{8.2}$$

Eq.(8.2) indicates that a parameter should learn upwards if CK+ samples at $s_i = (2.5+\Delta_a)(V)$, while learn downwards if CK+ samples at $s_i = (2.5 - \Delta_a)(V)$.

With the setting above, all training circuits of a CRBM system were tested individually for various levels of Δ_a. The measurement results in Fig.8.8 reveal that , if $|\Delta_D| = 2\Delta_a$ is smaller than a threshold value (e.g. 200mV), each training circuit updates its parameter in a fixed direction, either upwards or downwards, regardless of the direction suggested by Eq.(8.2). In addition, the fixed direction and the threshold value vary between training circuits. Therefore,

[3]Note that s_i and $\hat{s_i}$ share the same input in the training circuit of a CRBM system, and so do s_j and $\hat{s_j}$. The training circuit discussed here refers to training circuits for both $\{w_{ij}\}$ and $\{a_i\}$ s_j and $\hat{s_j}$ are simply replaced by s_i and $\hat{s_i}$ for training circuits for $\{a_i\}$.



(a) (b)

Figure 8.8: *(a)A measured w_{ij} learning upwards consistently despite of $\Delta_D = \Delta Y(1) = -203mV$ (b)A measured w_{ij} learning downwards consistently despite of $\Delta_D = \Delta Y(1) = +203mV$. Note that CKq=q1⊕q2⊕q3⊕q4 in this figure.*

each training circuit actually performs an MCD training rule as

$$\begin{aligned} \Delta\lambda &= \eta_\lambda \cdot sgn[\left(\langle s_i \cdot s_j \rangle_4 - \langle \hat{s}_i \cdot \hat{s}_j \rangle_4\right) + \Delta_T] \\ &= \eta_\lambda \cdot sgn[\Delta_D + \Delta_T] \end{aligned} \tag{8.3}$$

where λ represents any parameter of a CRBM system, and Δ_T denotes an "offset" existing in a training circuit. If $\Delta_T > |\Delta_D| > 0$, Eq.(8.3) indicates that the training circuit always implements an increase to λ, regardless of $\Delta_D < 0$. Conversely, if $\Delta_T < -|\Delta_D| < 0$, Eq.(8.3) indicates that the training circuit always implements a decrease to λ, regardless of $\Delta_D > 0$. Let Δ_{TW}, Δ_{TAV}, Δ_{TAH} denote the measured offsets for $\{w_{ij}\}$, the $\{a_i\}$ for visible neurons, and the $\{a_i\}$ for hidden neurons, respectively. The measured offsets are

$$\begin{aligned} \Delta_{TW} &= \begin{bmatrix} \times & -0.26 & -0.22 & +0.02 & -0.06 \\ +0.1 & -0.4 & +0.16 & -0.52 & -0.08 \\ -0.24 & -0.4 & -0.06 & +0.08 & -0.3 \end{bmatrix} (V) \\ \Delta_{TAV} &= \begin{bmatrix} +0.14 & +0.18 \end{bmatrix} (V) \\ \Delta_{TAH} &= \begin{bmatrix} +0.14 & 0 & -0.12 & +0.16 \end{bmatrix} (V) \end{aligned} \tag{8.4}$$

The result in Eq.(8.4) explains the measured malfunction of a training circuit in Fig.8.7. As w_{20} has an offset of -240mV, a contrastive divergence smaller than 200mV is not large enough to overcome this offset to update w_{20} in an increasing direction.

The offsets should be largely attributed to the accumulators of training circuits, in which dynamic current mirrors introduce errors by nonideal switching effects such as clock feedthrough [93]. Although accumulators can not be tested individually in the chips implemented in this research, the accumulators used in a CRBM system are identical to those used in [53]. The measurement results in [53] reveal that an accumulator introduces offsets with an average value of 7% of its input currents. As the reference-zero current for the accumulators in a CRBM system equals 2uA, the accumulators could introduce an average offset current of 0.14uA. According to Fig.6.2, the multiplier of a training circuit outputs 1uA when $s_i = s_j = 3.5V$ [4]. Overcoming an offset current of 0.14uA thus requires the input $(s_i - 2.5)$ to be greater than 0.14V when $s_j = 3.5V$. This estimation agrees with the measured offsets in Eq.(8.4), supporting the suggestion that accumulators are the circuits that introduce offsets to MCD training circuits.

8.3.2 Simulating the non-ideal training in a CRBM system

Eq.(8.3) indicates that nonzero offsets prevent training circuits from minimising contrastive divergences when $|\Delta_D| < |\Delta_T|$. In other words, the minimum divergence achievable for a training circuit is equivalent to its offset, $|\Delta_T|$. A CRBM system's inability to model training data faithfully in Sec.8.2 may therefore be attributed to the offsets existing in training circuits. To verify this explanation, a CRBM system was simulated in Matlab and was trained on the single-cluster data in Fig.8.4, with the training rule in Eq.(8.3) and the offsets given by Eq.(8.4). The model's learning rates (η) and noise variance(σ) were set to be the same as those of the

[4] As a multiplier is used in a training circuit, the inputs w_i and s_i of the multiplier correspond to the training circuit's inputs s_i and s_j, respectively.

CRBM system experimented in Sec.8.2.

The CRBM model was trained for 1000 epochs and the simulated traces of parameters are shown in Fig.8.9. Strikingly, Fig.8.9 agrees largely with the measurement result in Fig.8.5[5]. The slight mismatches are due to the fact that offsets actually depend on the input currents of accumulators, but each training circuit was assumed to introduce a constant offset in simulation. Therefore, the agreement between Fig.8.9 and Fig.8.5 supports the suggestion that offsets in training circuits are the primary non-ideal effect that inhibits a CRBM system from modelling data successfully.

The agreement between software simulations and hardware measurements further facilitates the simulation of a CRBM system's tolerance towards training offsets. To examine the maximum tolerable offset, it is assumed that the $|\Delta_T|$ for all parameters (Eq.(8.3)) are equivalent, whereas the signs of Δ_T for parameters remain in accord with Eq.(8.4). With various values of $|\Delta_T|$, a CRBM system's performance on modelling the same single-cluster data (Fig.8.4) was simulated in Matlab. Fig.8.10 shows the 20-step reconstructions generated by a CRBM after 4000 training epochs when $|\Delta_T|$ equals (a)0.05 (b)0.02 (c)0.01. As these values of $|\Delta_T|$ are smaller than the real measured values in Eq.(8.4), Fig.8.10 shows that the CRBM modelled one cluster of data points corresponding with the training data. However, the CRBM also generated extra data points around the top-left corner when $|\Delta_T|$ equals 0.05 and 0.02 (Fig.8.10(a) and (b)). This indicates that the offset values of 0.05 and 0.02 still cause several parameters to approach their limits. Such a nonideal effect finally disappears when $|\Delta_T|$ equals 0.01 (Fig.8.10(c))). Therefore, a CRBM can only tolerate a maximum offset of 0.01, even when modeling a dataset as simple as a single cluster of data points.

[5]Note that a higher value of $\{a_i\}$ in simulation corresponds to a lower voltage of $\{a_i\}$ in hardware, as indicated by Fig.6.7

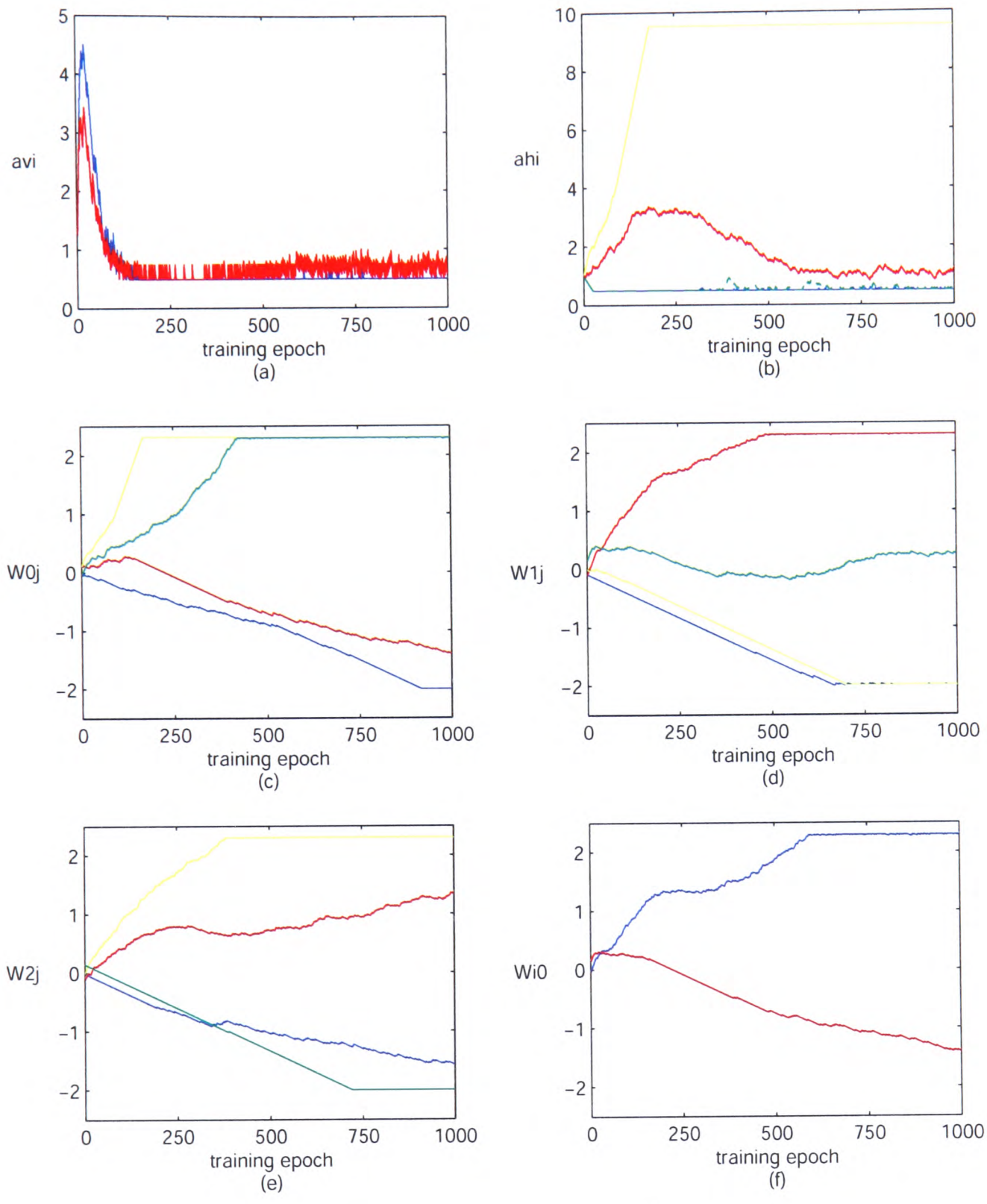


Figure 8.9: *The simulated traces of* $\{a_i\}$ *and* $\{w_{ij}\}$ *when a CRBM model was trained with the data in Fig.8.4 for 1000 epochs (a)*$\{a_{vi}\}$ *(b)*$\{a_{hi}\}$ *(c)*$\{w_{0j}\}$ *(d)*$\{w_{1j}\}$ *(e)*$\{w_{2j}\}$ *(f)*$\{w_{i0}\}$.

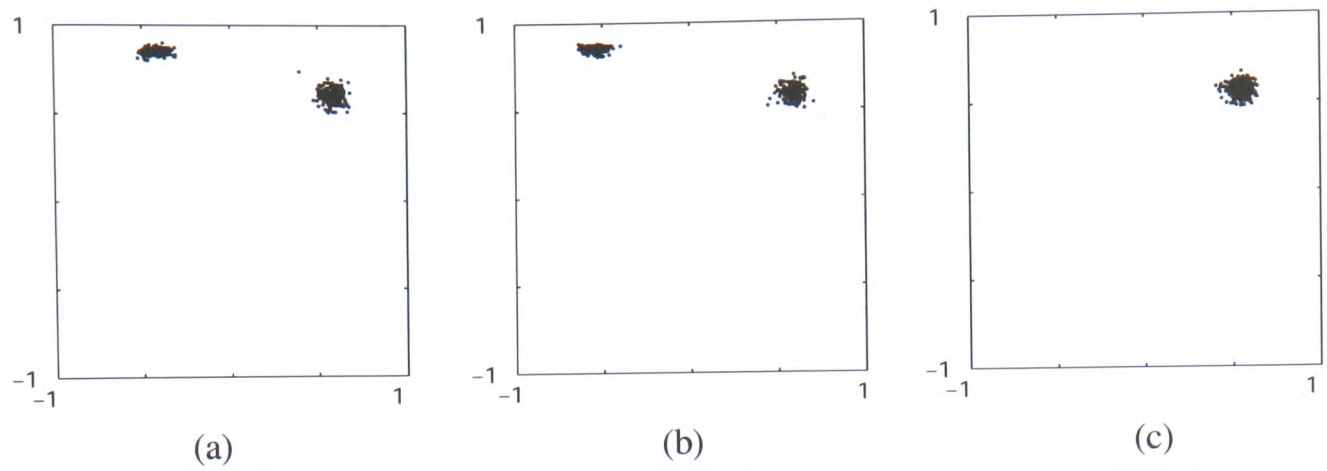


Figure 8.10: *The 20-step reconstructions generated by a CRBM after 4000 training epochs with the offset $|\Delta_T|$ in Eq.(8.3) equaling to (a)0.05 (b)0.02 (c)0.01.*

8.3.3 Discussion

As a unit contrastive divergence ($\Delta_D = 1$) is represented as 1uA in accumulators, an offset of 0.01 corresponds to an offset current as small as 10nA for accumulators. Such a small offset current is difficult to achieve with accumulators based upon dynamic current mirrors, because of nonideal switching effects[6]. Provided that the offset currents of accumulators equal to the previous estimated value 140nA and are independent of accumulators' input currents , it is possible to scale the current representing $\Delta_D = 1$ up to 15uA, for example, such that a tolerable offset of 0.01 corresponds to an offset current of 150nA. However, training circuits will subsequently consume 15 times more power, making a CRBM system impractically power-consuming. So an improved accumulator is essential for successful on-chip training in a CRBM system. It is notable that the discussion above is based on the assumption that offsets come primarily from accumulators. As training circuits implemented in this research do not allow their components to be tested individually, it is important to test this assumption in the future.

[6]10nA corresponds to a gate-voltage difference of merely 2.5mV for MOS transistors in accumulators of a CRBM system, while nonideal switching effects can easily introduce a gate-voltage error more than 2.5mV.

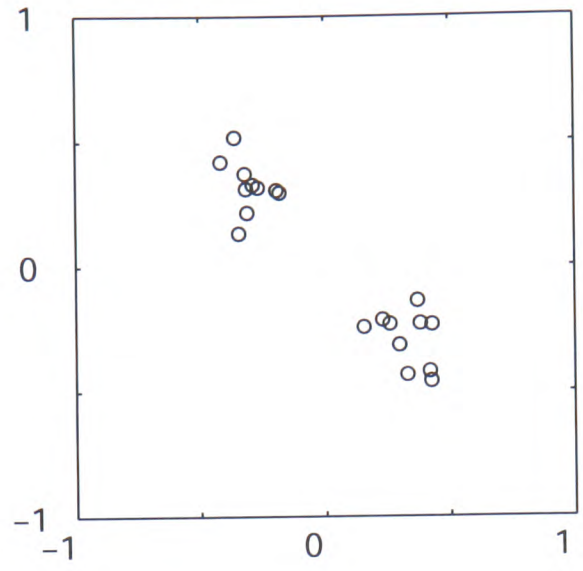


Figure 8.11: *20 training data points sampled from a distribution comprised of two circular Gaussians.*

8.4 Training a CRBM system with nonideal training circuits

In spite of non-negligible training offsets, a CRBM system's ability to model various continuous data was further inspected. $V_{\sigma v}$ and $V_{\sigma h}$ were set to be 4.05V, corresponding to $\sigma = 0.2$ in simulation (Fig.6.12). In an attempt to compensate training offsets a little, parameters were updated with carefully-chosen stepsizes. Let $+\eta_\lambda$ and $-\eta_\lambda$ denote the decreasing and increasing stepsizes, respectively, for a parameter λ. The training stepsizes for $\{w_{ij}\}$ and $\{a_i\}$ are $-\eta_{wij} = +\eta_{ahi} = 3mV$, $+\eta_{wij} = -\eta_{ahi} = +\eta_{avi} = 15mV$, and $-\eta_{avi} = -45mV$. The following subsections present and discusses experiment results of training a CRBM system on various continuous data.

8.4.1 Modelling data with a symmetric distribution

In the first experiment, a CRBM system was trained on the data shown in Fig.8.11 for 1000 epochs. The training data contain 20 data points sampled from a distribution comprised of two circular Gaussians. Fig.8.12 shows the measured traces of $\{a_i\}$ and $\{w_{ij}\}$ during training. Although most parameters still reached their limits after 1000 epochs, the CRBM system was

expected to model a distribution different from that in the experiment in Sec.8.2. The parameter values learnt at 50, 200, 500 and 1000 epoch were thus substituted into a CRBM model in Matlab, for generating the 20-step reconstructions displayed in Fig.8.13. The dotted lines in Fig.8.12 highlight parameter values corresponding to the reconstructions in Fig.8.13.

Fig.8.13(a) shows that the CRBM system found crude approximations to the training data at the onset of training. As several parameters reached their limits after 200 epochs, the variance of the reconstruction in Fig.8.13(b) decreases to as small as a single cluster in the training data. This change is mainly attributed to the fact that $\{a_{vi}\}$ reached their minimum value (i.e. maximum voltage 3V) after 200 epochs, as shown in Fig.8.12(a). This effectively reduces the output stochasticity of visible neurons, and thus the variance of the reconstruction. At epoch 500, the bias inputs for visible neurons, w_{10} and w_{20}, also reached their limits (Fig.8.12(f)), constituting a biasing vector $\mathbf{w}^{(0)} = \{w_{i0}\}$ pointing towards the bottom-right corner of the data space (Fig.8.14(b)). The reconstruction in Fig.8.13(c) therefore contains reconstructed points between the center and the bottom-right corner of the data space. Finally, as w_{14} increased to a positive value($> 2.5V$) and w_{24} remained negative($< 2.5V$) after 1000 epochs, the weight vector of hidden neuron H4, $\mathbf{w}^{(4)}$, approximately aligns with $\mathbf{w}^{(0)}$, encouraging data points to be reconstructed at the bottom-right corner as shown in Fig.8.13(d). Fig.8.13(d) reveals that the CRBM system finally modelled a distribution of two clusters, while training offsets prevent the CRBM system from modelling the correct positions of the clusters. Note that the variance of the bottom-right cluster is "squashed" by sigmoid functions.

Comparing this training experiment with that in Sec.8.2, Fig.8.14 shows the weight vectors of hidden neurons learnt in the two experiments. Although the traces of parameters in Fig.8.12 are very similar to those in Fig.8.5, Fig.8.14 reveals that the weight vectors learnt by hidden neurons H2 and H4 after 1000 epochs differ significantly in the two experiments. As training offsets caused the bias vector $\mathbf{w}^{(0)}$ to point towards the bottom-right corner in both

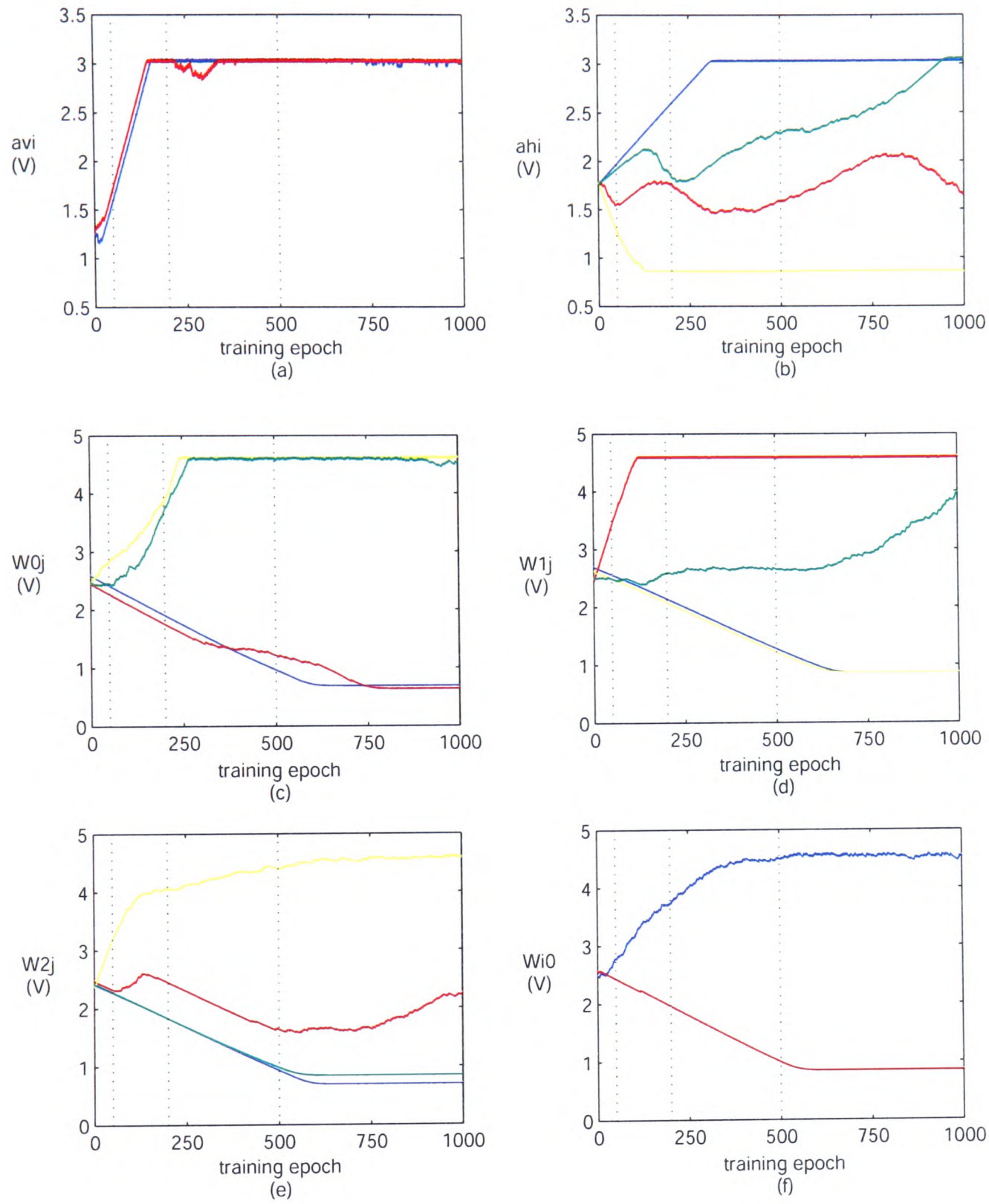


Figure 8.12: *The measured traces of* $\{a_i\}$ *and* $\{w_{ij}\}$ *when a CRBM system was trained with the data in Fig.8.11 for 1000 epochs (a)*$\{a_{vi}\}$ *(b)*$\{a_{hi}\}$ *(c)*$\{w_{0j}\}$ *(d)*$\{w_{1j}\}$ *(e)*$\{w_{2j}\}$ *(f)*$\{w_{i0}\}$*. The grey dotted lines highlight the parameter values used to generate 20-step reconstructions in Fig.8.6.*

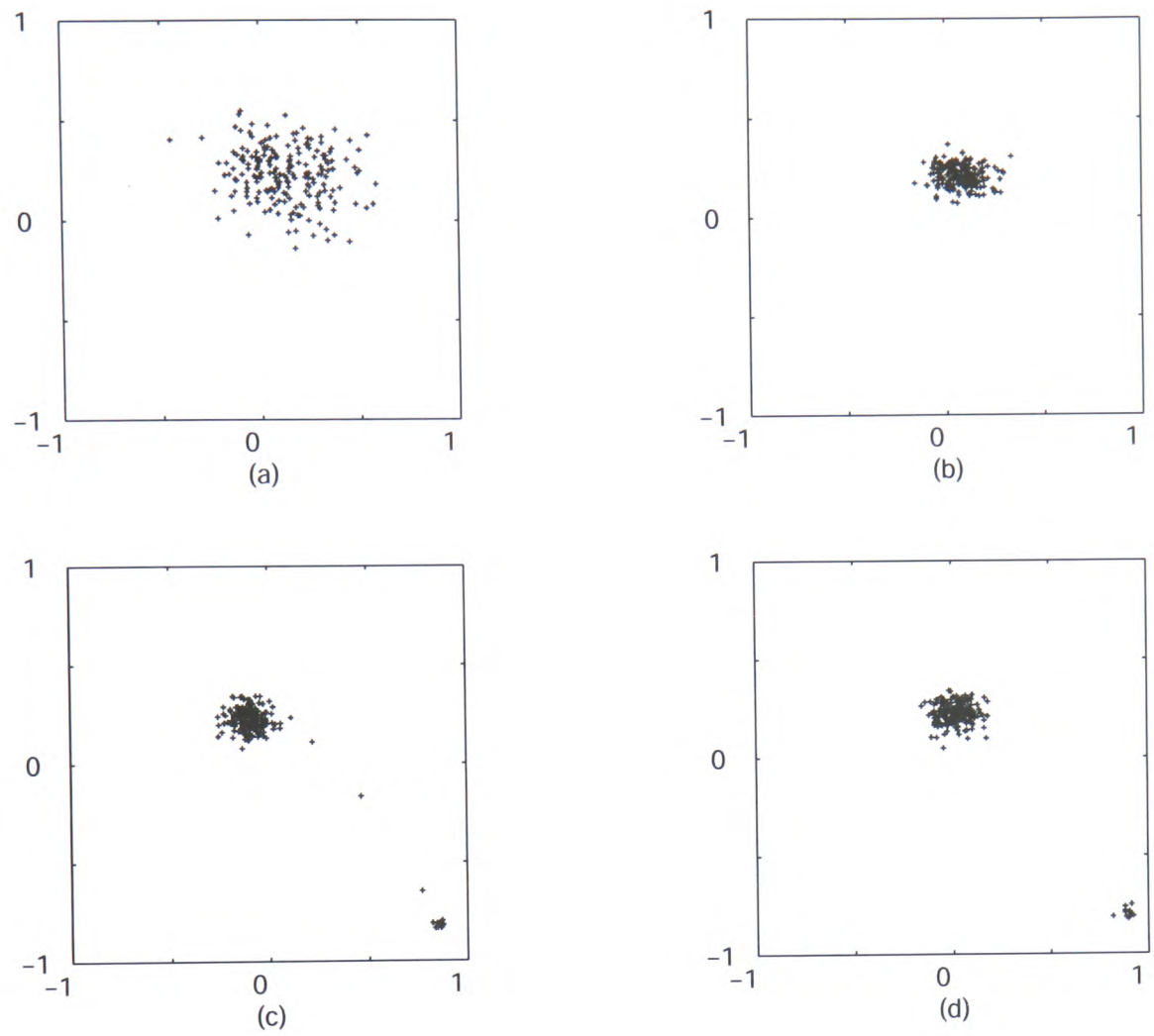


Figure 8.13: *The 20-step reconstructions generated by a CRBM model in Matlab with parameters values learnt by a CRBM system after (a)50 (b)200 (c)500 (d)1000 epochs*

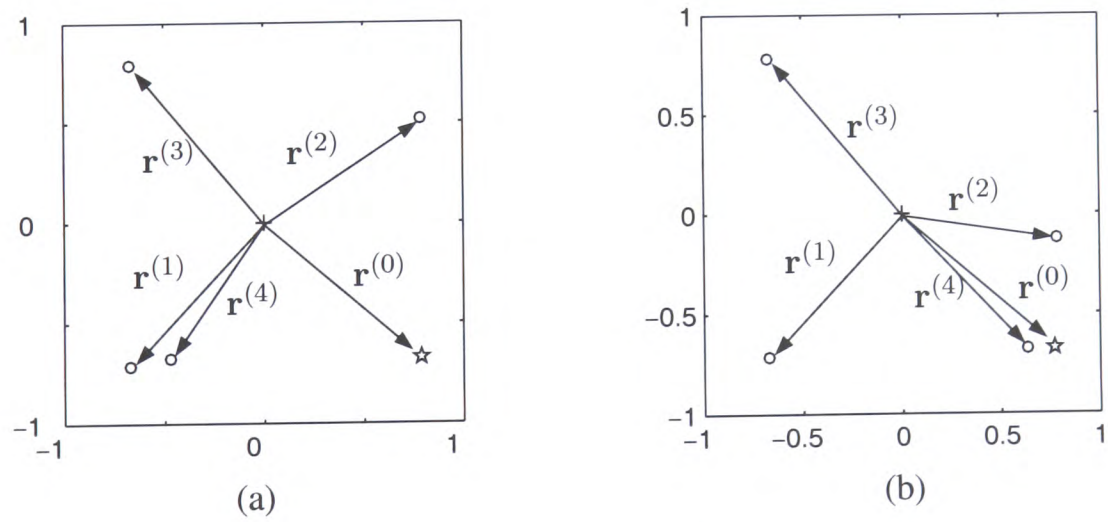


Figure 8.14: *Projections of the trained weight vectors of hidden neurons in data space, where* $\mathbf{r}^{(i)} = \phi(\mathbf{w}^{(i)})$ *and* $\mathbf{w}^{(i)}$ *represents the weight vector of hidden neuron* i *(a)experiment with single-cluster training data in Fig.8.4 (b)experiment with two-cluster training data in Fig.8.11.*

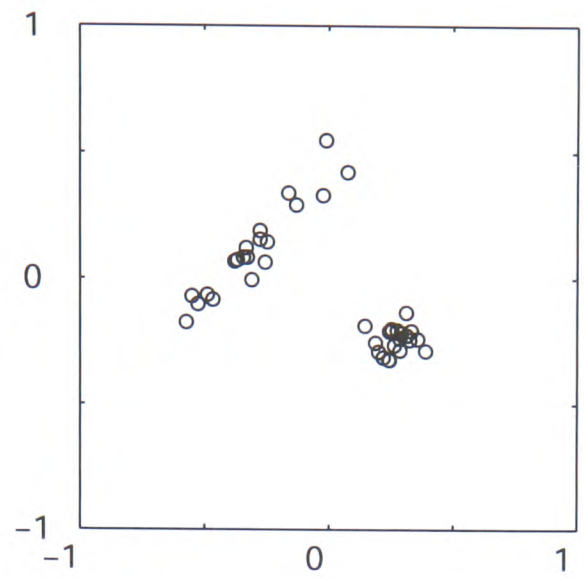


Figure 8.15: *20 training data points sampled from a distribution comprises one elliptic Gaussian and one circular Gaussian.*

experiments, Fig.8.14(a) indicates that the CRBM system adapted $\mathbf{w}^{(2)}$ and $\mathbf{w}^{(4)}$ to discourage reconstructions at the bottom-right corner, i.e. to assign a low probability to the bottom-right corner. The reconstruction in Fig.8.6(d) therefore contains only one cluster of data points. On the contrary, Fig.8.14(b) shows that the CRBM system adapted $\mathbf{w}^{(2)}$ and $\mathbf{w}^{(4)}$ to encourage reconstructions at the bottom-right corner, resulting in a reconstruction of two clusters of data points (Fig.8.13(d)). The evident difference between two experiments demonstrates a CRBM system's adaptability and attempt to minimise contrastive divergences despite training offsets.

8.4.2 Modelling data with a non-symmetric distribution

To test a CRBM system's maximum learning capability, a CRBM system was trained on data with a more complicated distribution as shown in Fig.8.15. The training data contains 20 data points sampled from a distribution comprised of one elliptic and one circular Gaussians. Fig.8.16 shows the measured traces of $\{a_i\}$ and $\{w_{ij}\}$ during 1000 training epochs. As a_{v1} displays a dramatic change between epoch 300 and epoch 800, the parameters values learnt at epoch 50, 200, 400, 500, 660, and 1000 were substituted into a CRBM in Matlab for generating

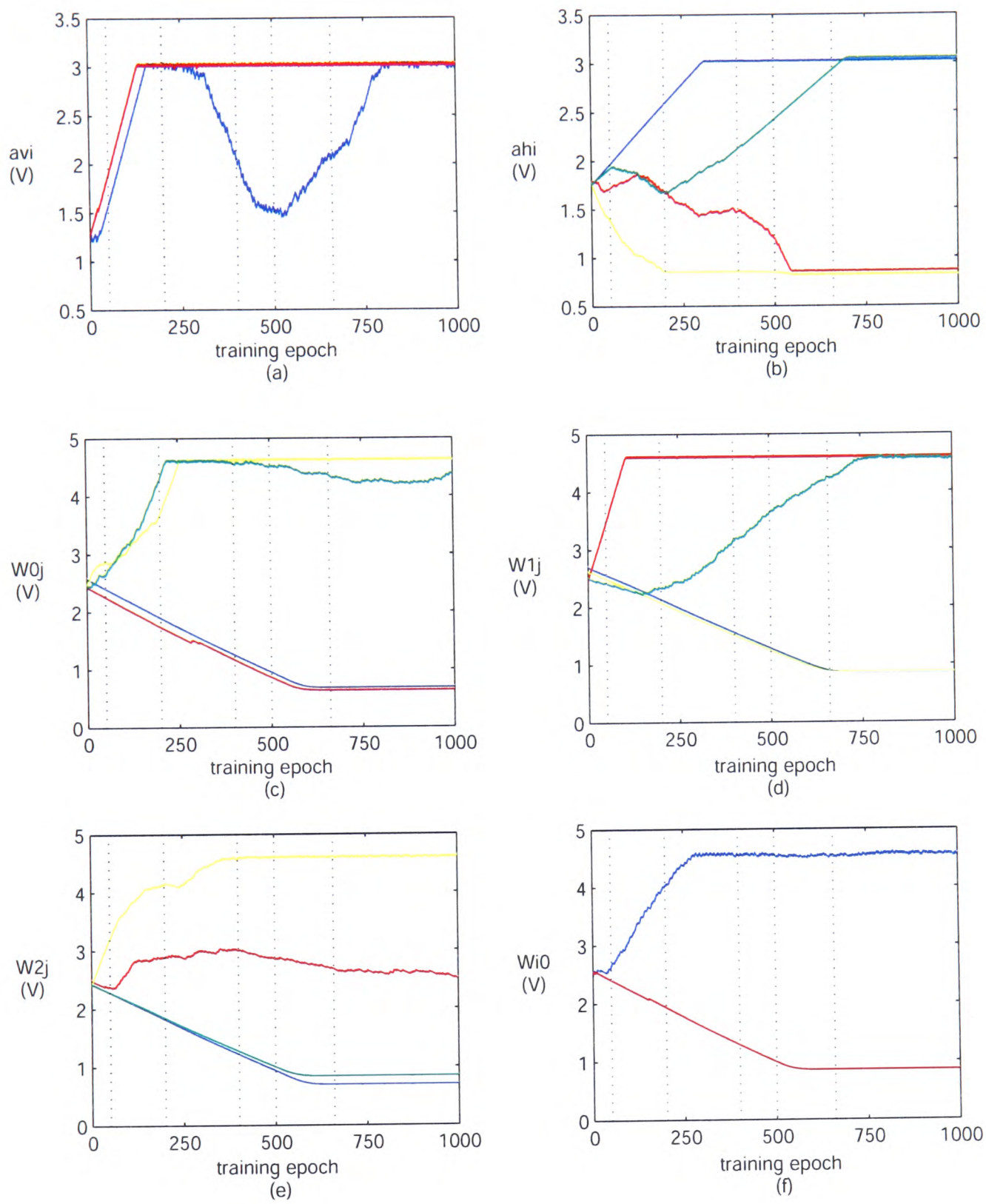


Figure 8.16: *The measured traces of* $\{a_i\}$ *and* $\{w_{ij}\}$ *when a CRBM system was trained with the data in Fig.8.15 for 1000 epochs (a)*$\{a_{vi}\}$ *(b)*$\{a_{hi}\}$ *(c)*$\{w_{0j}\}$ *(d)*$\{w_{1j}\}$ *(e)*$\{w_{2j}\}$ *(f)*$\{w_{i0}\}$.

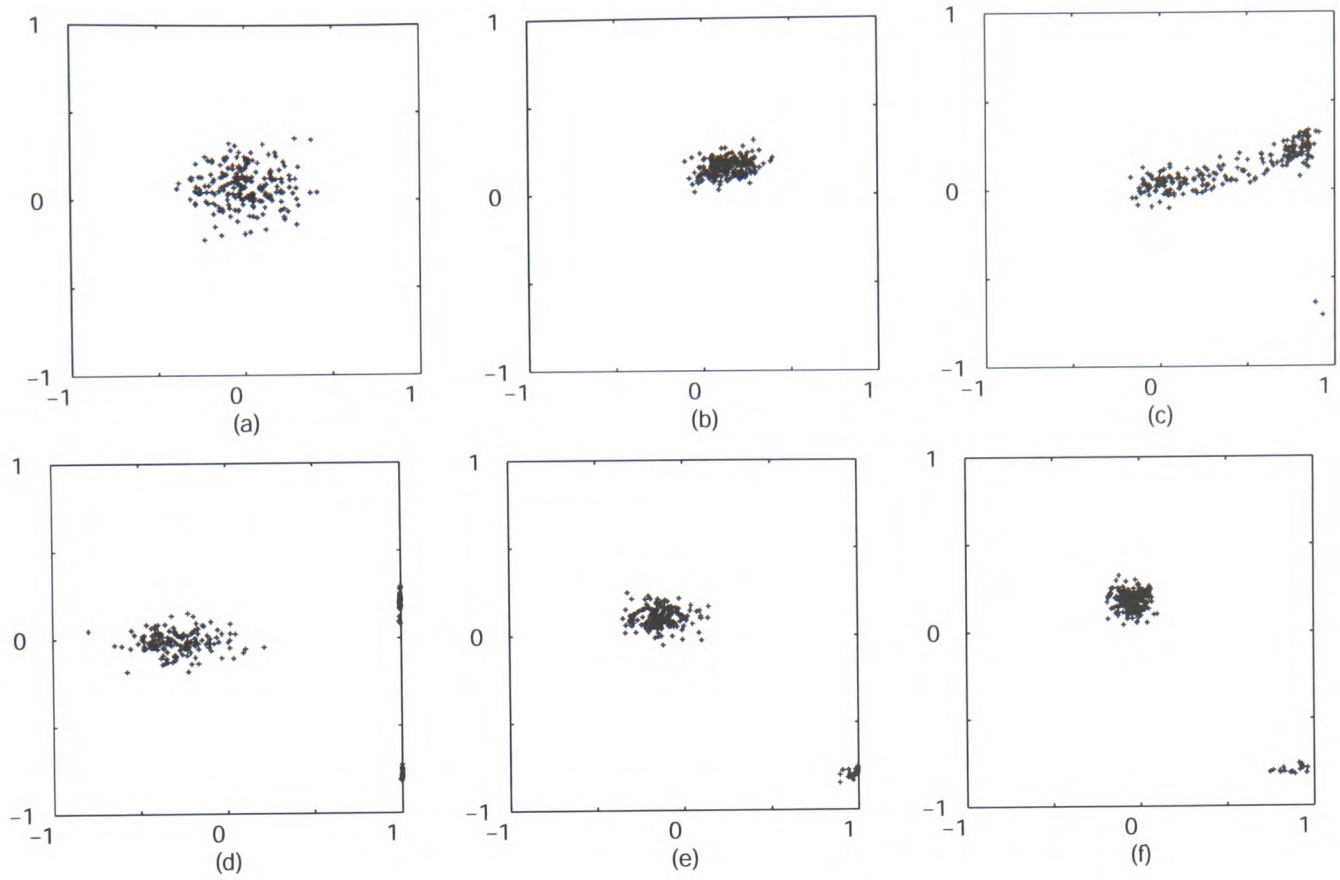


Figure 8.17: *The 20-step reconstructions generated by a CRBM model with parameters values learnt by a CRBM system after (a)50 (b)200 (c)400 (d)500 (e)660 (f)1000 epochs*

the 20-step reconstructions in Fig.8.17.

Fig.8.17(a) and (b) show that the CRBM system's performance was similar to that in previous experiments in the first 200 epochs. However, the training data in this experiment exhibit a large variance in the dimension corresponding to visible neuron V1. The CRBM system thus decreased a_{v1} after epoch 200, attempting to represent the variance roughly by increasing output stochasticity of V1. Fig.8.17(c) shows that this effort results in a reconstruction of an elliptic cluster of data points at epoch 400. However, more parameters reached limits at epoch 500, causing reconstructed points to occur on the boundary of the data space (Fig.8.17(d)). To compensate for these problems, the CRBM system increased a_{v1} back towards a higher value (Fig.8.16(a)), reducing the output stochasticity of V1. Fig.8.17(e) reveals that the reconstruction at epoch 660 consequently contains a reduced number of data points on the boundary. Finally, as a_{v1} reached its limit after 1000 epochs, no reconstructed point occurs on the bound-

ary (Fig.8.17(f)), whereas the modelled distribution is similar to that in Fig.8.13(d), instead of the training data in Fig.8.15.

This experiment indicates that the fabricated CRBM system has too few adaptable parameters to model a data distribution as complicated as Fig.8.15, although the CRBM system displayed an "effort" to model training data before most parameters reached their limits. Improved training circuits are therefore essential for achieving satisfactory on-chip training.

8.5 Reconstructing continuous data distributions

To explore the utility of continuous-valued probabilistic behaviour in VLSI, a CRBM system's ability to reconstruct continuous data distributions is tested. Although nonideal training offsets discourage on-chip training, a good mapping between hardware and software allows the training of a CRBM system to be simulated in Matlab as in Ch.5, and learnt parameters can then be "downloaded" onto a CRBM system for reconstructing modelled distributions. To simulate a CRBM system as faithfully as possible, the Matlab simulations in this section take into account all hardware simplifications and limitations discussed in Ch.5, except for the training offsets measured in Sec.8.3.

Fig.8.18 shows the measurement setup for generating multi-step reconstructions from a CRBM system. With the internal digital circuits of a CRBM system disabled, one DAQ board generates the digital signals (CKdata, CKV, and CKH) to sample the neurons of a CRBM system for multiple steps. The other DAQ board then presents random initial data to visible neurons, and records the outputs of visible neurons during multi-step reconstructions. With the voltage comparators producing refresh signals for parameters, the function generator clocks CKup to refresh parameters at a frequency of 10kHz. Finally, noise inputs for neurons are generated from function generators. The following subsections present and discuss the measure-

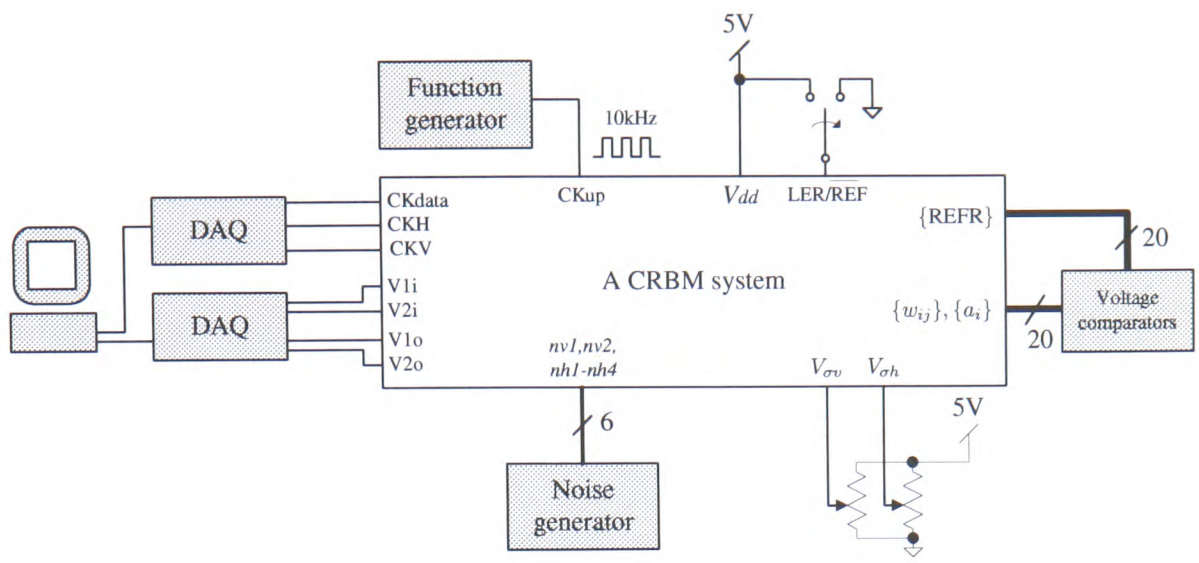


Figure 8.18: *The measurement setup for carrying out multi-step reconstruction in a CRBM system.*

ment results of using a CRBM system to reconstruct a variety of continuous data distributions.

8.5.1 Reconstructing a single, circular cluster of data points

With $\eta_w = \eta_{ah} = 0.015$, and $\eta_{av} = 0.09$[7], a CRBM model was trained on the single-cluster data in Fig.8.19(a) for 3000 epochs. The 20-step reconstruction generated by the trained CRBM model is shown in Fig.8.19(b). According to the mapping for $\{a_i\}$ in Fig.6.7 and with the reference-zero for $\{w_{ij}\}$ shifted to 2.5(V), the voltage values corresponding to the parameter values learnt by the CRBM model are

$$\begin{aligned}
\{w_{ij}\} &= \begin{bmatrix} \times & 3.452 & 2.612 & 3.064 & 0.961 \\ 2.916 & 2.569 & 2.738 & 2.849 & 2.219 \\ 2.484 & 3.033 & 2.313 & 2.441 & 1.959 \end{bmatrix} (V) \\
\{a_{vi}\} &= \begin{bmatrix} 1.737 & 1.520 \end{bmatrix} (V) \\
\{a_{hi}\} &= \begin{bmatrix} 2.275 & 1.326 & 1.456 & 1.241 \end{bmatrix} (V)
\end{aligned} \tag{8.5}$$

[7] η_{av} and η_{ah} denote the learning rates for the $\{a_i\}$ for visible and hidden neurons, respectively

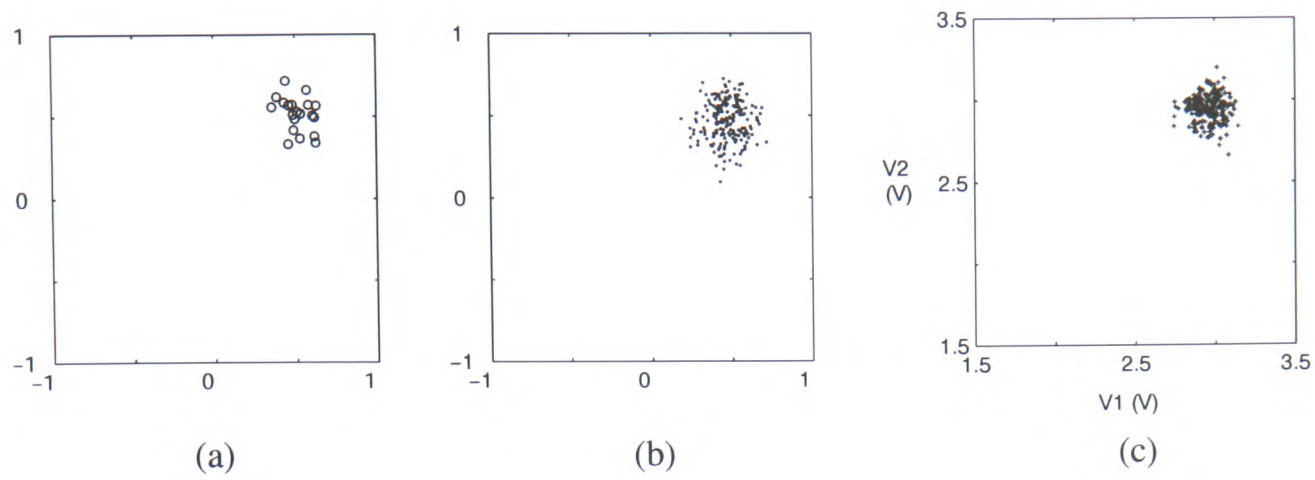


Figure 8.19: *(a)Training data sampled from a single, circular Gaussian (b)The 20-step reconstruction generated by a CRBM model after being trained on the data in (a) for 3000 epochs (c)The 20-step reconstruction generated by a CRBM system with parameters refreshed to the levels in Eq.(8.5)*

With $\{w_{ij}\}$ and $\{a_i\}$ refreshed to the levels given in Eq.(8.5), a CRBM system generated the 20-step reconstruction as shown in Fig.8.19(c), from 200 random initial data. Fig.8.19(c) shows that the CRBM system reconstructed the training data successfully. The match between Fig.8.19(b) and Fig.8.19(c) furthermore demonstrates that parameter values in a CRBM model were correctly converted into voltage values in a CRBM system.

8.5.2 Reconstructing data with a symmetric distribution

To demonstrate the probabilistic behaviour of a CRBM system more clearly, a CRBM system was tested to reconstruct the training data in Fig.8.20(a), containing two circular clusters of data points. A CRBM model was trained on this data with $\eta_w = \eta_{ah} = 0.003$, and $\eta_{av} = 0.03$ for 30000 epochs. Fig.8.20(b) shows the 20-step reconstruction generated by the trained CRBM model, and the learnt parameter values correspond to voltage values of

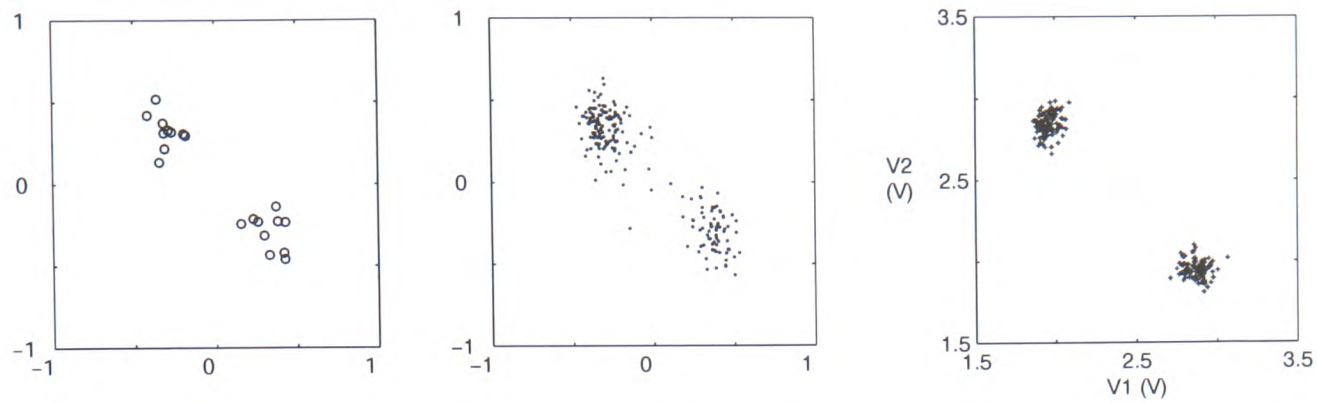


Figure 8.20: *(a)Training data sampled from two circular Gaussians. (b)The 20-step reconstruction generated by a CRBM model after being trained on the data in(a) for 30000 epochs (c)The 20-step reconstruction generated by a CRBM system with parameters refreshed to the levels in Eq.(8.6)*

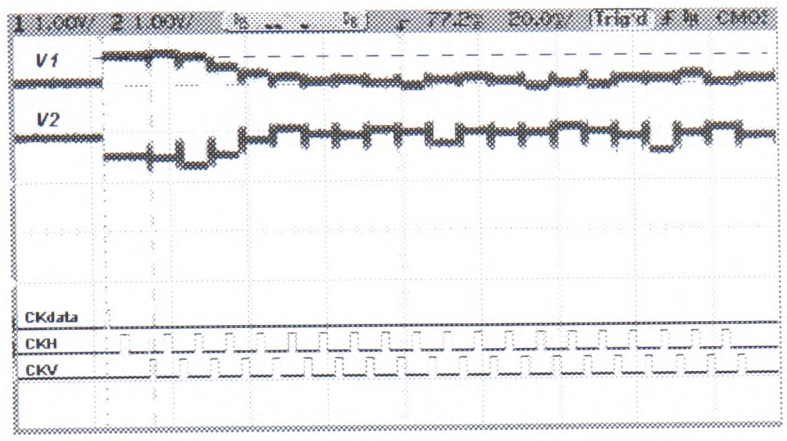
$$\{w_{ij}\} = \begin{bmatrix} \times & 3.203 & 2.598 & 2.400 & 1.319 \\ 2.862 & 2.280 & 2.180 & 4.061 & 2.589 \\ 2.616 & 2.365 & 2.471 & 1.811 & 2.565 \end{bmatrix} (V) \tag{8.6}$$

$$\{a_{vi}\} = \begin{bmatrix} 2.458 & 1.651 \end{bmatrix} (V)$$

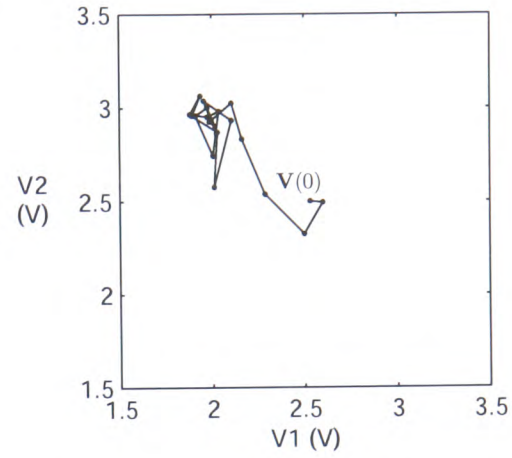
$$\{a_{hi}\} = \begin{bmatrix} 1.503 & 2.516 & 1.286 & 1.551 \end{bmatrix} (V)$$

With $\{w_{ij}\}$ and $\{a_i\}$ refreshed to the levels given in Eq.(8.6), the CRBM system generated the 20-step reconstruction as shown in Fig.8.20(c). The separation between the two clusters in Fig.8.20(c) is slightly greater than those in Fig.8.20(a)(b). As Ch.4 has shown that the separation between clusters results primarily from a large value of $\{a_i\}$ for hidden neurons, this slight mismatch is attributed to the unavoidable mapping error for $a_{h3} = 1.286V$, which was derived according to Fig.6.7. Nevertheless, the approximate match between Fig.8.20(c) and Fig.8.20(a)(b) demonstrates again that the CRBM system reconstructed the training data successfully.

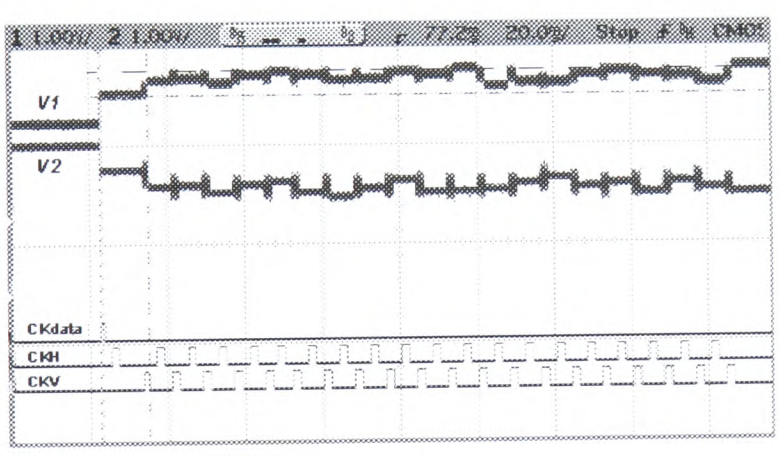
Moreover, the "continuous-valued probabilistic dynamics" in a CRBM system was explored. Fig.8.21 shows the measured activities of the visible neurons when the CRBM system



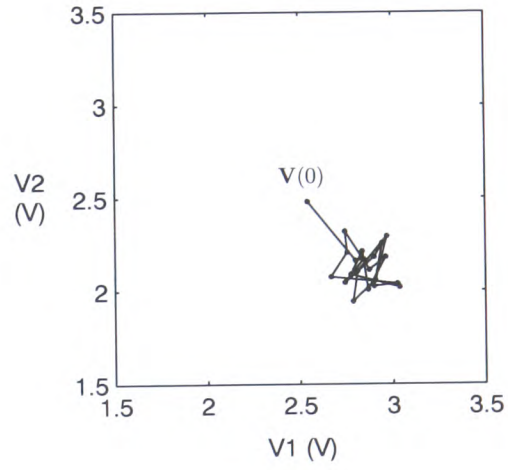
(a)



(b)



(c)



(d)

Figure 8.21: *The measured activities of visible neurons during Gibbs sampling from a CRBM system for 20 steps (a)Visible states move from initial state (2.5,2.5)(V) towards the top-left cluster (b)Trace composed of the sampled visible states in (a). (c)Visible states move from initial state (2.5,2.5)(V) towards the bottom-right cluster (d)Trace composed of the sampled visible states in (c).*

were Gibbs sampled for 20 steps. Fig.8.21(a) and (c) display two opposite measurement results, each of which includes the measured outputs of the visible neurons, V1 and V2, during 20-step Gibbs sampling and the digital signals that control the sampling process. In each measurement, the visible neurons were first set to a constant initial state, $\mathbf{V}(0) = (2.5, 2.5)(V)$, and then sampled for 20 steps. Each measurement therefore contains 20 sampled visible states. Fig.8.21(b) and (d) plots the sampled visible states, in Fig.8.21(a) and (c) respectively, into the state space of the visible neurons , revealing the traces of the visible states during 20-step Gibbs sampling.

Fig.8.21(b) and (d) show that, although the initial visible state $\mathbf{V}(0)$ equals (2.5,2.5)(V) in both measurements, the Gibbs-sampled visible states can move either towards the top-left cluster or towards the bottom-right cluster step by step. The visible states reach one of the clusters in the first few steps, and continue to move around the region corresponding to this cluster. The formation of the two-cluster reconstruction in Fig.8.20(c) thus becomes clear. Furthermore, this measurement result demonstrates clearly the continuous-valued probabilistic "dynamics" of a CRBM system. Imagine the "dynamics" of the visible state as a ball moving in the two-dimensional visible-state space. The noise in neurons prevents the visible state from stabilising into any particular state, while the parameters of the CRBM system encode the regions with higher probabilities, which correspond to two attractors in this experiment, encouraging visible states to move towards and then around these regions.

8.5.3 Reconstructing data with a non-symmetric distribution

A CRBM system was further tested to reconstruct a more complicated distribution which comprises data points sampled from one elliptic Gaussian and one circular Gaussian, as shown in Fig.8.22(a). After 30000 training epochs with $\eta_w = \eta_{ah} = 0.003$, and $\eta_{av} = 0.03$, a CRBM model generated the 20-step reconstruction shown in Fig.8.22(b). The learnt parameter values correspond to voltage values of

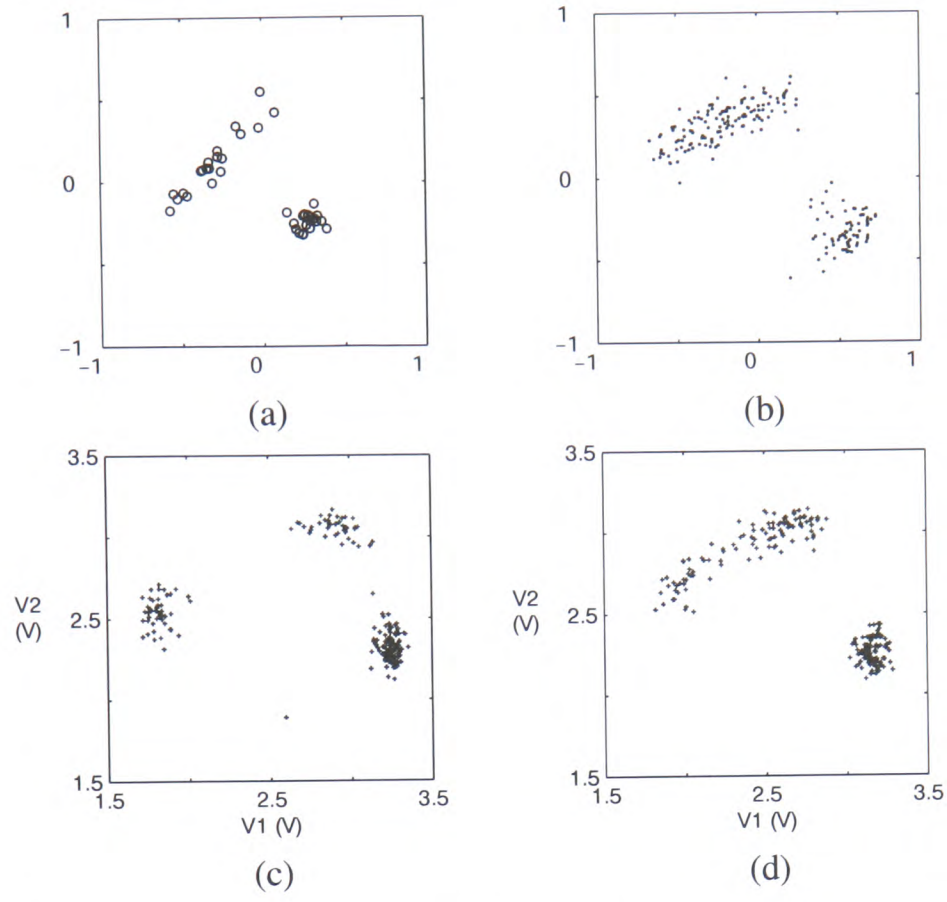


Figure 8.22: *(a)Training data sampled from one elliptic and one circular Gaussians. (b)The 20-step reconstruction generated by a CRBM model after being trained on the data in(a) for 30000 epochs (c)The 20-step reconstruction generated by a CRBM system with parameters refreshed to the levels in Eq.(8.7) (d)The 20-step reconstruction generated by a CRBM system after a_{v1}, a_{h1}, and w_{10} are adjusted to 2.1V, 2.274V, and 2.45V, respectively*

$$
\begin{aligned}
\{w_{ij}\} &= \begin{bmatrix} \times & 2.662 & 2.702 & 1.927 & 2.425 \\ 2.571 & 4.544 & 1.661 & 2.446 & 2.748 \\ 2.186 & 3.732 & 3.875 & 2.451 & 2.593 \end{bmatrix} (V) \\
\{a_{vi}\} &= \begin{bmatrix} 2.011 & 2.151 \end{bmatrix} (V) \\
\{a_{hi}\} &= \begin{bmatrix} 1.8287 & 1.2083 & 2.0883 & 2.4835 \end{bmatrix} (V)
\end{aligned}
\tag{8.7}
$$

With $\{w_{ij}\}$ and $\{a_i\}$ refreshed to the levels given in Eq.(8.7), the CRBM system generated the 20-step reconstruction as shown in Fig.8.22(c). The cluster at the bottom-right corner in Fig.8.22(c) corresponds to the circular cluster in training data. However, the reconstruction in Fig.8.22(c) corresponds with the elliptic cluster in training data roughly by another two

separate clusters. As discussed in Sec.8.5.2, mismatches such as the separation of clusters are largely attributed to the unavoidable errors of mapping $\{a_i\}$ from software to hardware. With a_{v1} adjusted to 2.1V, a_{h1} to 2.274V, and w_{10} to 2.45V, a reconstruction agreeing with the training data, as shown in Fig.8.22(d), was consequently obtained. This experiment implies that reconstructing a complicated distribution requires a more sophisticated tuning of parameter values. This requirement however is likely to become trivial, provided the CRBM system can be trained on-chip to achieve a minimum-divergence solution.

8.5.4 Reconstructing data with a doughnut-shaped distribution

To highlight the distinctive continuous-valued probabilistic behaviour a CRBM system possesses, a CRBM system was used to reconstruct data with a doughnut-shaped distribution. With $\eta_w = 0.00075$, $\eta_{av} = \eta_{ah} = 0.0075$, a CRBM model was trained on the data in Fig.8.23(a) for 20000 epochs[8]. The trained CRBM model generated the 20-step reconstruction as shown in Fig.8.23(b). Although the reconstruction in Fig.8.23(b) also forms a circular band, the reconstruction has a larger variance than the training data. This is attributable to the fact that four hidden neurons provide a limited number of parameters for representing the circular band well in different angles. The learnt parameter values correspond to voltages as follows

$$\begin{aligned}
\{w_{ij}\} &= \begin{bmatrix} \times & 2.503 & 2.548 & 2.451 & 2.498 \\ 2.497 & 2.964 & 2.514 & 2.526 & 2.025 \\ 2.498 & 2.973 & 2.503 & 2.434 & 2.975 \end{bmatrix} (V) \\
\{a_{vi}\} &= \begin{bmatrix} 1.287 & 1.294 \end{bmatrix} (V) \\
\{a_{hi}\} &= \begin{bmatrix} 1.154 & 1.835 & 1.898 & 1.173 \end{bmatrix} (V)
\end{aligned} \tag{8.8}$$

With $\{w_{ij}\}$ and $\{a_i\}$ refreshed to the levels given in Eq.(8.8), the CRBM system generated the 20-step reconstruction as shown in Fig.8.23(c). As in the experiment in Sec.8.5.3, this initial

[8] As four hidden neurons provides limited number of parameters for modelling a doughnut-shaped distribution, hardware simplifications and limitations were not taken into account in this simulation

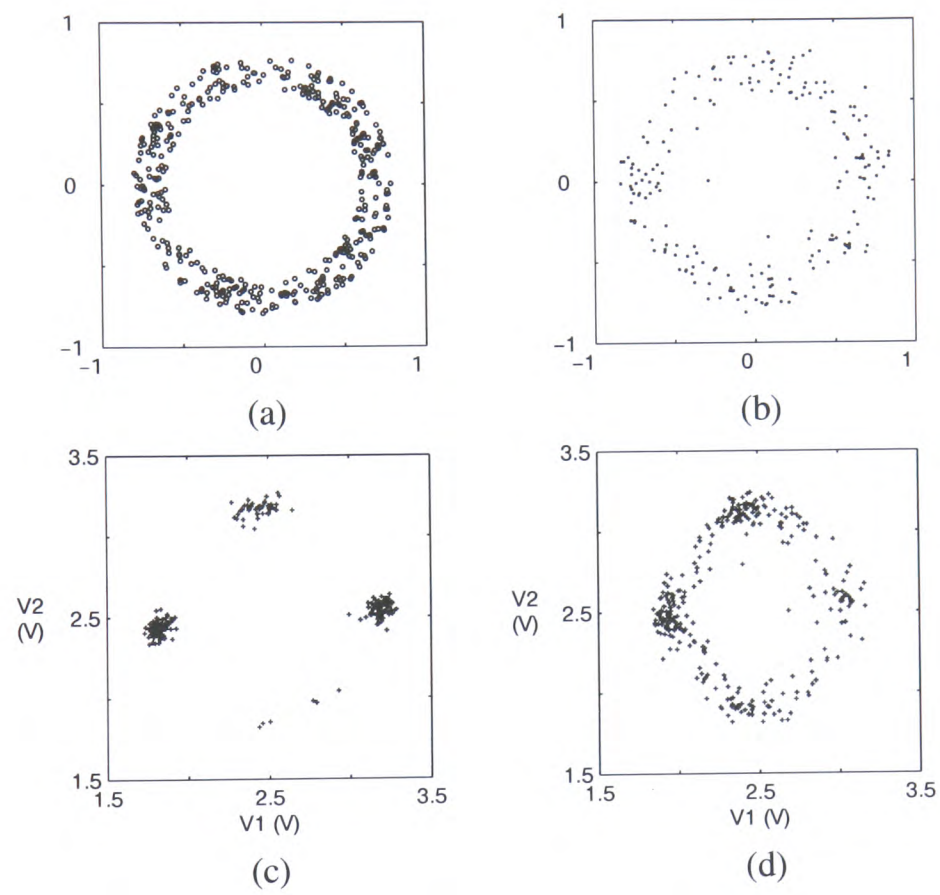
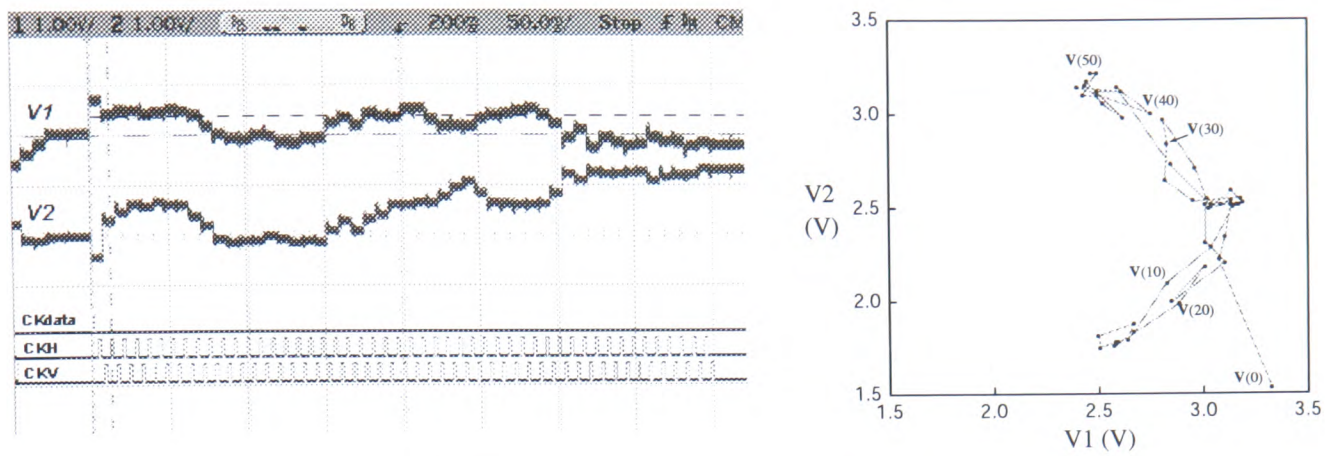


Figure 8.23: *(a)Training data with a doughnut-shaped distribution. (b)The 20-step reconstruction generated by a CRBM model after being trained on the data in(a) for 20000 epochs (c)The 20-step reconstruction generated by a CRBM system with parameters refreshed to the levels in Eq.(8.8) (d)The 20-step reconstruction generated by a CRBM system after a_{v1} and a_{v2} are adjusted to 1.41V and 1.1V, respectively.*



(a) (b)

Figure 8.24: *(a)The measured activities of visible neurons during 50-step Gibbs sampling when a doughnut-shaped distribution is modelled by the CRBM system (b)The trace composed of the visible states in (a).*

reconstruction possesses separate clusters of data instead of data points evenly-distributed in a circular band. An improved reconstruction, as shown Fig.8.23(d), is thus obtained simply by adjusting $a_{v1} = 1.41\text{V}$ and $a_{h1} = 1.1\text{V}$. Although the reconstruction in Fig.8.23(d) retains a slightly uneven distribution, it is notable that the reconstruction by a CRBM model (Fig.8.23(d)) has a similar uneven distribution. The uneven distribution is thus mainly attributable to the limited number of hidden neurons.

Fig.8.24(a) shows the measured activities of the visible neurons when the CRBM system was Gibbs sampled for 50 steps. Fig.8.24(b) shows the trace composed of the 50 sampled visible states in Fig.8.24(a). The initial visible state $\mathbf{V}(0)$ corresponds to a point at the bottom-right corner of the state space. Fig.8.24(b) shows that the visible state moves into the circular band of the doughnut-shaped distribution in the first few steps, and then visits different states in the right half of the circular band with nearly-equal probability. This demonstrates the "dynamics" of a CRBM system as a more sophisticated, stochastic moving of its visible state. Imagine that the *energy landscape*[9] defined by the CRBM system is a circular band of a low-energy valley

[9]The energy landscape defined by a generative model is inversely proportion to the probabilistic distributions defined by the model, as explained in Ch.2.

in the visible-state space, and that the visible state is a ball moving in the valley. Fig.8.24(b) shows that the visible state travels along the right half of the valley, but is "bounced" back at the top and bottom ends of the circular band. This indicates that the CRBM system creates small energy barriers in these places, If the visible state does not get enough "noise" (energy) when it is visiting these places, it can not jump over the barriers to visit the left half of the circular band. The slight uneven reconstruction in Fig.8.23(d) thus becomes clear. The utility and richness of noise-induced, continuous-probabilistic behaviour in VLSI is finally highlighted.

8.6 Summary

A CRBM system containing two visible and four hidden neurons has been implemented. The training experiments show that a CRBM system is able to adapt its continuous-valued probabilistic behaviour towards minimising contrastive divergence. However, the training circuits are shown to exhibit training offsets, which prevent the CRBM system from achieving a minimum divergence, i.e. from modelling training data optimally. By modelling the training offsets in Matlab simulation, it is shown that a CRBM can only tolerate a maximum offset of 0.01, even when modelling a dataset as simple as a single cluster of data points. This indicates that improved training circuits are essential for achieving fully satisfactory on-chip training in a CRBM system. Moreover, this reflects that training a probabilistic model still requires precise calculations, even though probabilistic computation potentially possesses tolerance towards computational errors. Finally, the utility of continuous-valued probabilistic behaviour in VLSI is demonstrated as a CRBM system's ability to reconstruct a variety of continuous data distributions. The continuous-valued probabilistic "dynamics" is also demonstrated in VLSI, highlighting the distinctive probabilistic behaviour a CRBM system possesses.

Chapter 9
Summary and Conclusion

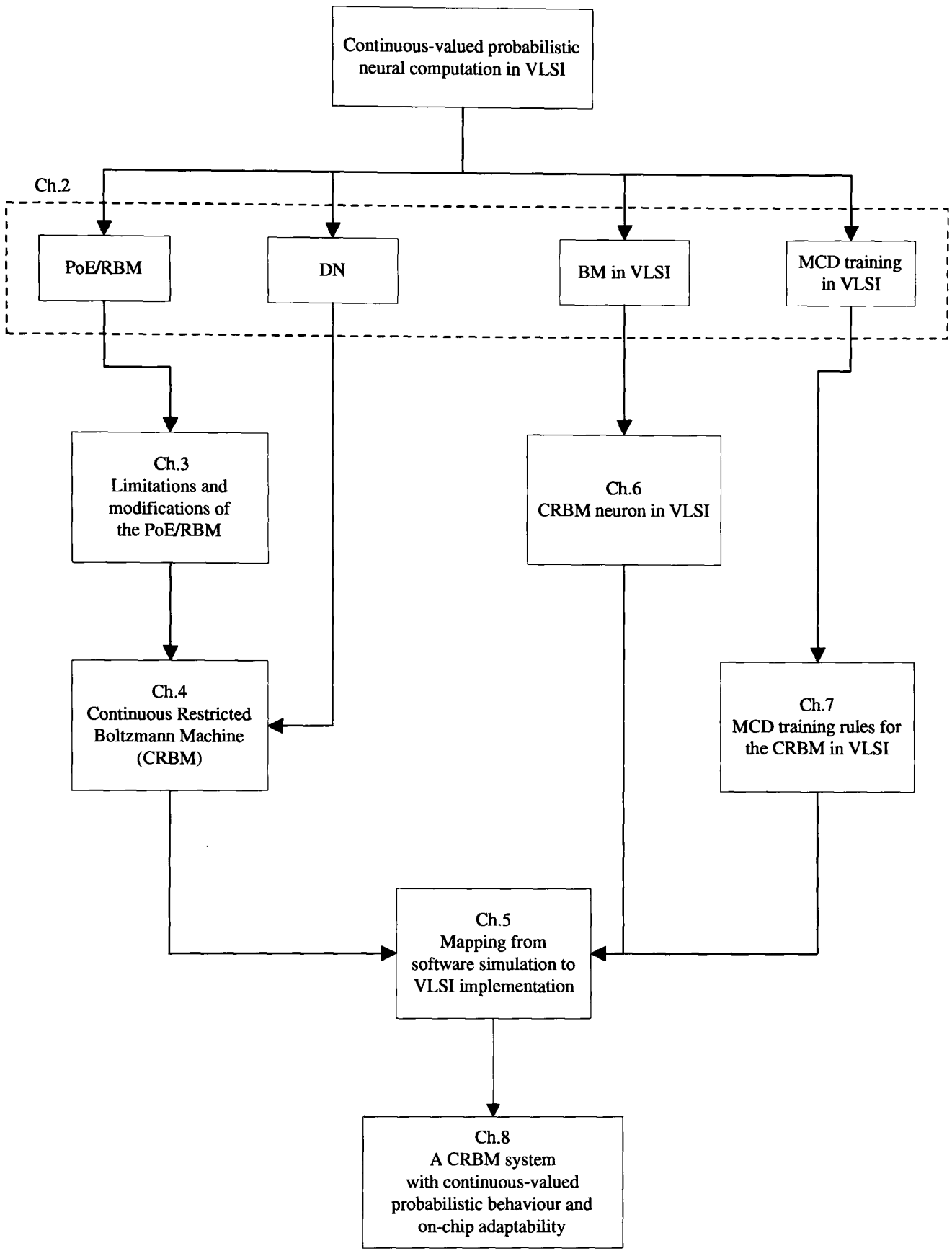
9.1 Summary

Fig.9.1 illustrates the entire research process in this thesis, revealing how it builds on two separate lines of research, one algorithmic and the other in hardware, and finally converges on the "demonstrator" for this project, a VLSI system with continuous-valued probabilistic behaviour and on-chip adaptability.

In the algorithm domain, experiments with artificial data show that the PoE/RBM only generates data with a symmetric distribution and is unable to model an arbitrary continuous-valued probability density faithfully. The PoE/RBM therefore is not optimal for an intelligent embedded system. Although RBMrate is shown to be able to alleviate the limitations of the PoE/RBM, the repetitive sampling of an RBMrate makes it unsuitable for VLSI implementation.

The *Continuous Restricted Boltzmann Machine* (CRBM) is developed by substituting continuous stochastic neurons with Gaussian noise inputs into the "restricted" architecture of the PoE/RBM. The training algorithm for the CRBM was derived by applying the Minimising-Contrastive-Divergence (MCD) method to the training rules of the Diffusion Network. With a small approximation, the training algorithm requires only multiplication and subtraction and thus facilitates hardware-amenability. Experiments with artificial data show that the CRBM can model continuous data successfully with this simple, reliable training algorithm. Experiments with real heartbeat data further confirm that the CRBM is able to model complex data.

In the hardware domain, based on the Boltzmann-Machine neuron in VLSI [19, 39, 50],



BM : Boltzmann Machine
DN : Diffusion Network
PoE/RBM : the Products of Experts in RBM form
MCD : Minimising-Contrastive-Divergence

Figure 9.1: *The flowchart illustrating the research process in this thesis.*

the CRBM neurons have been implemented in VLSI. A wide-range four-quadrant multiplier accommodates weight inputs ranging from 0V to V_{dd}, and an improved sigmoid circuit introduces an adaptable, near-ideal sigmoidal nonlinearity. With artificially-generated noise, CRBM neurons in VLSI are shown to exhibit the continuous-valued probabilistic behaviour that this project aims to explore.

Based on the architecture of the MCD training circuit proposed in [52, 53], the MCD training circuits for the CRBM have also been implemented. A pulse-controlled learning circuit was designed not only to enable parameters to span the full range of a power supply, but also to ease the setting of learning rates. Results show that the MCD training circuit can adapt a parameter with variable stepsizes and in a correct direction. The training stepsize can be as small as 2.5mV, facilitating the minimisation of training errors introduced by hardware simplifications and limitations.

As a detailed mapping from the CRBM algorithm into a CRBM system has been derived, the two lines of research converge to the "demonstrator" for this project, a VLSI system with continuous-valued probabilistic behaviour and on-chip adaptability. The CRBM system provides a platform for studying both the on-chip adaptability and the utility of noise-induced, continuous-valued probabilistic behaviour in VLSI. Training experiments show that a CRBM system can adapt its parameters towards minimising contrastive divergences. However, training circuits on chip are shown to exhibit training offsets, which prevent a CRBM system from modelling training data optimally. By modelling the training offsets in Matlab simulation, it was found that a CRBM can only tolerate a maximum training offset of 0.01 (in numeric value) even when modelling a data set as simple as a single cluster of data points. Finally, the utility of noise-induced, continuous-valued probabilistic behaviour in VLSI is demonstrated as a CRBM system's ability to reconstruct a variety of continuous-valued data distributions.

9.2 Conclusion

This thesis examined the suggestion that

"*Continuous-valued probabilistic behaviour can be realised effectively in VLSI circuits*".

From development of the CRBM, this research has demonstrated that (a) neurons with noise-induced stochasticity, (b) a "restricted" architecture, and (c) MCD training method can be "fused" to enhance the representational ability and the hardware-amenability of a continuous-valued probabilistic model. Neurons with noise-induced stochasticity not only remove the limitations and artefacts introduced by binary stochastic neurons, but also provide the ability to develop diverse stochastic behaviour in the CRBM. This leads to a modelling flexibility that exceeds the ability of models with more fixed-form (e.g. binary) neurons. This is all clear from the experiments with artificial data designed to probe how a CRBM forms a generative model. It is further demonstrated that the restricted architecture can enhance the distinctive functions of neurons in different layers (the visible and the hidden layers). The weight vectors of hidden neurons have been shown to encode meaningful and useful data features in many experiments. This provides the advantage that hidden neurons can underpin a simple, reliable novelty detector, as demonstrated in experiments with ECG data. Moreover, it has been demonstrated that the CRBM can be trained reliably by the MCD method. The MCD training algorithm is proved to be inexpensive in both software and hardware implementation. The CRBM is therefore a promising generative model that is both useful and hardware-amenable, and it provides a basis for exploring continuous-valued probabilistic behaviour in VLSI.

From the VLSI implementation of the CRBM neurons, this research has demonstrated that the incorporation of artificially-generated noise is a simple and effective way to introduce continuous-valued probabilistic behaviour to VLSI. Noise signals can be added to deterministic signals in the current mode, and the continuous-valued probabilistic nature of noise can be preserved in VLSI circuits without difficulty. This encourages the use of noise-induced,

continuous-valued probabilistic behaviour in VLSI for computation. However, it was found that noise generators have the potential to interfere with analogue references, introducing extra computational errors (e.g.imprecise learning rates). Careful implementation of noise sources is therefore necessary to "localise" noise generation to circuits requiring probabilistic behaviour.

From the training experiments with a CRBM system, this research further demonstrated that noise-induced, continuous-valued probabilistic behaviour in VLSI can be adapted on-chip towards modelling a continuous-valued probabilistic distribution. However, it was found that a CRBM system demanded precise training circuits to achieve satisfactory on-chip training. This result reflects an important fact. The tolerance towards environmental interferences and computational errors relies on the existence of probabilistic behaviour. Such tolerance therefore holds only for computation with "stochastic" neurons, not for a "deterministic" training circuit, in a CRBM system. However, the noise generator and digital circuits in a CRBM system inevitably distribute noise to other circuits and increase computational errors. This points towards the need for a technique that allows both *noisy* and *noise-free* circuits to co-exist in the same VLSI chip, in order to achieve a probabilistic VLSI system with reliable on-chip adaptability.

Furthermore, this research has demonstrated that noise-induced, continuous-valued probabilistic behaviour in VLSI can be utilised to reconstruct a variety of continuous data distributions. Measurement results further confirm that continuous-valued dynamics of a probabilistic model can be realised effectively and demonstrated in VLSI. This encourages the use of noise in VLSI for computation, which refers specifically to representing the variability of continuous data in a CRBM system. In addition, it was found that a CRBM system was sensitive to parameter setting when reconstructing complicated data distributions. This indicates that on-chip adaptability is essential for a probabilistic model in VLSI , such as a CRBM system, to reach an optimum setting (modelling) in the existence of process variations. It is expected that a CRBM system with improved training circuits will be suitable for an intelligent embedded system that

can adapt its "internal noise" to generalise and thus to tolerate the noise in biomedical signals.

Finally, it is concluded that, as this thesis suggested, continuous-valued probabilistic behaviour can be realised effectively in VLSI, although this research has not solved all of the detailed problems in order to make such a system perfect.

9.3 Future work

As the promising modelling ability and the hardware-amenability of the CRBM have been demonstrated, it is encouraged to further explore the capabilities of the CRBM in more biomedical applications, and to realise a CRBM-based, intelligent embedded system. Computation with noise-induced probabilistic behaviour in VLSI, furthermore, point towards several interesting fields to be investigated.

The signals in general biomedical applications are likely to be high-dimensional and more complicated than those examined in this thesis. For example, an implantable instrument usually incorporates a large number of sensors to maximise information and to enhance its reliability. These sensory signals are thus high-dimensional data, not the two-dimensional data used in many of the experiments in this research. Moreover, sensory or biomedical signals may contain multiple categories which are not linearly separable. It is therefore important to further explore the CRBM's ability to model high-dimensional data and to solve nonlinear problem.

Modelling high-dimensional data further brings out the need for an efficient way to evaluate the performance of a CRBM. While the performance of a CRBM has been evaluated by visually comparing multi-step reconstructions, it is difficult to compare high-dimensional reconstructions in the same way. An efficient and reliable way of estimating the fitness of a CRBM model to a high-dimensional probability density is thus vital.

The CRBM system implemented in this research is a "demonstrator" for studying the feasibility and the utility of continuous-valued probabilistic computation in VLSI. To become an intelligent embedded system, the power- and area- consumption of a CRBM system must be further optimised. In addition, an improved training circuit is essential for achieving a satisfactory on-chip adaptability. A scalable design and programmable connections between neurons are also important to achieve a large network of neurons in the future. This will facilitate further exploration of continuous-valued probabilistic computation in VLSI, as well as the use of a CRBM system in real biomedical applications.

The training experiments with a CRBM system have reflected an important and interesting field to be investigated, which is an effective way to "localise" noise generation in a VLSI chip. This is vital not only to enhance noise-induced probabilistic behaviour, but also to minimise computational errors in a CRBM system. If this is achievable, a CRBM system could become a noisy but robust system.

The research in this thesis further leads to a new thought on intrinsic VLSI noise. While intrinsic VLSI noise is normally suppressed as much as possible to enhance the precision of circuits, the successful development of a CRBM system opens up the possibility of utilising intrinsic VLSI noise for computation. The computation would then refer to using "intrinsic noise" to enhance a system's "intelligence" and robustness against external noise or computational errors. Intrinsic VLSI noise could thus become an advantage rather than an obstacle in VLSI circuits. This is especially attractive in the deep-sub-micron technology where noise in transistors will grow dramatically.

Finally, the similarity between the CRBM and the Diffusion Network suggests that the Diffusion Network should be also realisable in VLSI. A Diffusion Network in VLSI could then provide a platform for further exploring continuous-valued, probabilistic dynamics in VLSI. For

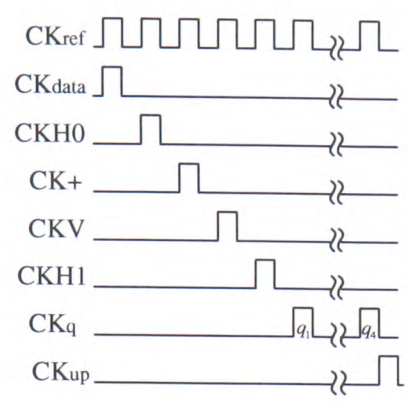
example, [28] has derived an algorithm to train the dynamics of a diffusion process to model distributions of data sequences. This means that the state of a Diffusion-Network system could be further trained to visit specific traces in its continuous state space. This feature corresponds to a more sophisticated control over noise-induced, probabilistic behaviour in VLSI, and would lead continuous-valued probabilistic computation in VLSI to a broader range of applications.

Appendix A
Digital-control circuits for training a CRBM system

Fig.A.1(a) redraws the digital-control signals in Fig.5.6(a) by dividing CKH into CKH0 and CKH1, as well as combining q1-q4 into a signal CKq. CKH0 and CKH1 sample initial and one-step Gibbs-sampled hidden states, respectively, and CKq controls the accumulation of four contrastive divergences. Fig.A.1(a) reveals that the digital-control signals consist of non-overlapping clocks and follow a simple turning-on sequence. The first six clocks (from CKdata to CKq) are thus able to be implemented by a 3-flip-flop state machine as shown in Fig.A.2. The outputs of the flip-flops are fed back to their inputs according to

$$\begin{aligned} S_0^{t+1} &= S_0 \cdot S_1 + S_1 \cdot \overline{S_2} \\ S_1^{t+1} &= \overline{S_0} \cdot S_2 + S_1 \cdot \overline{S_2} \\ S_2^{t+1} &= S_0 \cdot S_1 + \overline{S_1} \cdot S_2 \end{aligned} \tag{A.1}$$

As clocked by CKref, such feedback connections cause the flip-flips to change their states in accord with the sequence in Fig.A.1(b). Since the sequence simply circulates through six different combination states, each combination state can correspond to one of the six clocks by tapping the outputs of the flip-flops to the AND gates as



(a)

t			t+1		
S_0	S_1	S_2	S_0	S_1	S_2
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	0	0	1

(b)

Figure A.1: *(a)The digital-control signals for training a CRBM system (b)The state-changing sequence for a 3-flip-flop state machine to generate the clock signals in (a)*

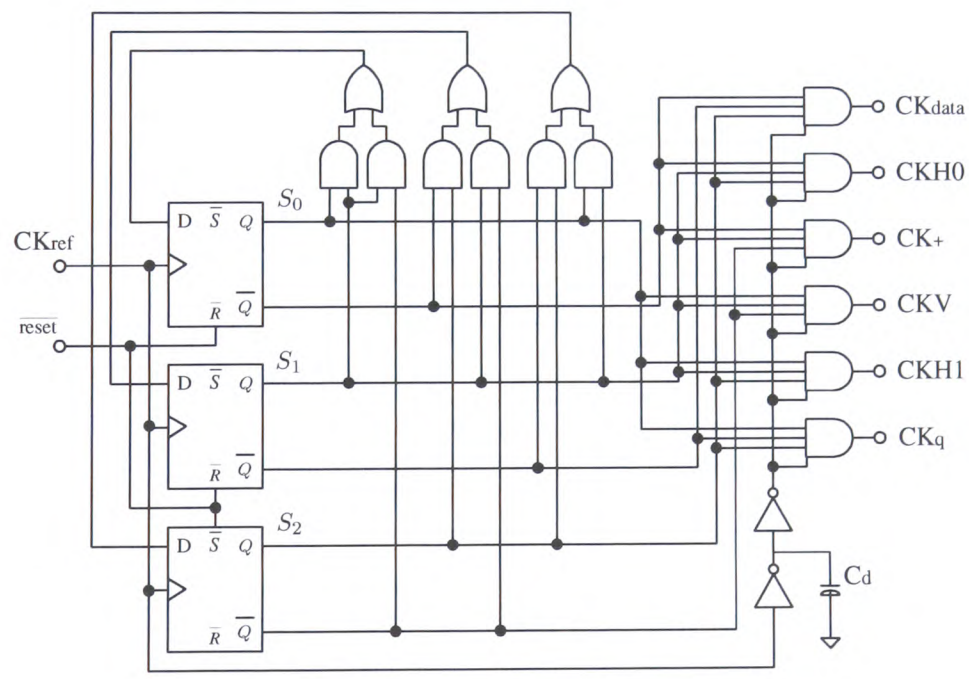


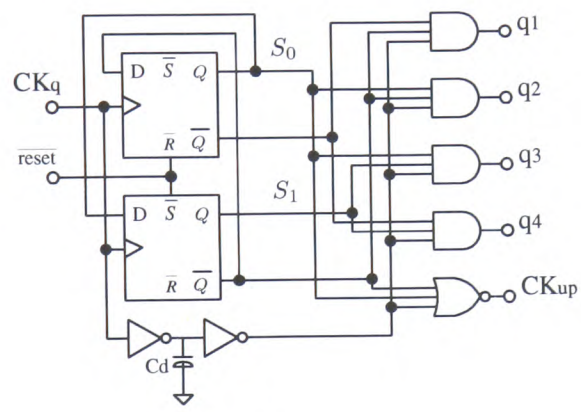
Figure A.2: *The digital circuit that generates control signals for training a CRBM system*

$$\begin{aligned}
\text{CKdata} &= \overline{S_0} \cdot \overline{S_1} \cdot S_2 \cdot \text{CKref} \\
\text{CKH0} &= \overline{S_0} \cdot S_1 \cdot S_2 \cdot \text{CKref} \\
\text{CK+} &= \overline{S_0} \cdot S_1 \cdot \overline{S_2} \cdot \text{CKref} \\
\text{CKV} &= S_0 \cdot S_1 \cdot \overline{S_2} \cdot \text{CKref} \\
\text{CKH1} &= S_0 \cdot S_1 \cdot S_2 \cdot \text{CKref} \\
\text{CKup} &= S_0 \cdot \overline{S_1} \cdot S_2 \cdot \text{CKref}
\end{aligned} \tag{A.2}$$

The incorporation of CKref synchronises the generated clocks with CKref.

Similarly, the four clocks for current accumulators, q1-q4, are able to be generated from CKq with a 2-flip-flop state machine, as illustrated in Fig.A.3(a). The feedback connections of the flip-flops follow $S_0^{t+1} = S_1$ and $S_1^{t+1} = \overline{S_0}$, causing the flip-flops to change their states in accord with the sequence in Fig.A.3(b). The four clocks q1-q4, as well as the updating clock (CKup) for learning circuits, are thus generated according to

$$\begin{aligned}
\text{q1} &= \overline{S_0} \cdot \overline{S_1} \cdot \text{CKq} \\
\text{q2} &= S_0 \cdot \overline{S_1} \cdot \text{CKq} \\
\text{q3} &= S_0 \cdot S_1 \cdot \text{CKq} \\
\text{q4} &= \overline{S_0} \cdot S_1 \cdot \text{CKq} \\
\text{CKup} &= \overline{S_0} \cdot S_1 \cdot \overline{\text{CKq}} = \overline{S_0 + \overline{S_1} + \text{CKq}}
\end{aligned} \tag{A.3}$$



(a)

t		t+1	
S_0	S_1	S_0	S_1
0	0	1	0
1	0	1	1
1	1	0	1
0	1	0	0

(b)

Figure A.3: *(a)The digital circuit that generates q1-q4 for accumulators and CKup for learning circuits. (b)The state-changing sequence for the 2-flip-flop state machine in (a)*

Appendix B
Publication list

This appendix lists the papers published as the contributions of this PhD research:

Chen, H., Fleury, P., and Murray, A.F., "Computation with Noise-induced, Continuous-valued Probabilistic Behaviour in a VLSI generative model," submitted to *IEEE transactions on Neural Networks*.

Chen, H. and Murray, A.F., "How approximate to Continuous Boltzmann Equilibrium is a Continuous Restricted Boltzmann Machine," submitted to *Neural Networks*

Chen, H., Fleury, P., and Murray, A.F., "Unsupervised Probabilistic Neural Computation in Mixed-mode VLSI," in *Smart Adaptive Systems on Silicon* (M. Valle ed..) Kluwer Academic. *In Press*.

Chen, H., Fleury, P., and Murray, A.F., "Minimizing Contrastive Divergence in Noisy, Mixed-mode VLSI Neurons," in *Advances in Neural Information Processing Systems (NIPS2003)*, vol.16, In Press, 2004.

Chen, H. and Murray, A.F., "A Continuous Restricted Boltzmann Machine with an Implementable Training Algorithm," in *IEE Proceedings of Vision, Image and Signal Processing*. Vol.150, no.3, p153-158, 2003.

Tang, T.B., Chen, H., and Murray, A.F., "Adaptive, Integrated Sensor Processing to Compensate for Drift and Uncertainty: a Stochastic 'Neural' Approach," in *IEE Proceeding on Nanobiotechnology*, Vol.151, no.1, p28-34, 2004.

Chen, H., Fleury, P., Tang, T.B., and Murray, A.F., "Adaptive Noisy Neural Computation in Mixed-mode VLSI," in *Proceedings of the Seventh International Conference on Cognitive and Neural Systems*, p.68, (Boston, USA), May 2003.

Fleury, P., Chen, H., Murray, A.F. "On-chip Contrastive Divergence Learning in Analogue VLSI," in *Proceedings of the International Joint Conference on Neural Networks*, In Press, 2004.

Tang, T.B., Chen, H. and Murray, A.F., "Adaptive Stochastic Classifier for Noisy pH-ISFET Measurements," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN2003)*, p638-645, 2003.

Chen, H. and Murray, A.F., "A Continuous Restricted Boltzmann Machine with a Hardware-Amenable Learning Algorithm," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN2002)* p358-363, 2002.

References

[1] T. B. Tang, E. Johannessen, L. Wang, A. Astaras, M. Ahmadian, A. F. Murray, J. M. Cooper, S. P. Beaumont, B. W. Flynn, and D. R. S. Cumming, "Towards a miniature wireless integrated multisensor microsystem for industrial and biomedical applications," *IEEE Sensors Journal: Special Issue on Integrated Multisensor Systems and Signal Processing*, vol. 2, no. 6, pp. 628–635, 2002.

[2] E. A. Johannessen, L. Wang, L. Cui, T. B. Tang, M. Ahmadian, A. Astaras, S. W. Reid, P. Yam, A. F. Murray, B. W. Flynn, S. P. Beaumont, D. R. S. Cumming, and J. M. Cooper, "Implementation of distributed sensors in a microsystems format," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 3, pp. 525–535, 2004.

[3] K. Najafi and K. D. Wise, "An implantable multielectrode array with on-chip signal processing," *IEEE Journal of Solid-State Circuits*, vol. 21, no. 6, pp. 1035 – 1044, 1986.

[4] T. W. Berger, M. Baudry, R. D. Brinton, J.-S. Liaw, V. Z. Marmarelis, A. Yoondong Park, B. J. Sheu, and A. R. Tanguay, "Brain-implantable biomimetic electronics as the next era in neural prosthetics," *Proceedings of the IEEE*, vol. 89, no. 7, pp. 993 – 1012, 2001.

[5] K. D. WISE, D. J. ANDERSON, J. F. HETKE, D. R. KIPKE, and K. NAJAFI, "Wireless implantable microsystems: high-density electronic interfaces to the nervous system," *Proceedings of the IEEE*, vol. 92, no. 1, pp. 76 – 97, 2004.

[6] M. Sun, M. Mickle, L. Wei, Q. Liu, and R. Sclabassi, "Data communication between brain implants and computer," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 11, no. 2, pp. 189–192, 2003.

[7] M. A. L. Nicolelis, "Actions from thoughts," *Nature*, vol. 409, pp. 403–407, Jan. 2001.

[8] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, p. 109, 118 1990.

[9] I. C. Jou, R. Y. Liu, and C. Y. Wu, "Cmos implementation of neural networks for speech recognition," in *IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 513–518, 1994.

[10] N. Aibe, M. Yasunaga, I. Yoshihara, and J. H. Kim, "A probabilistic neural network hardware system using a learning-parameter parallel architecture," in *Proceedings of the 2002 International Joint Conference on Neural Networks*, vol. 3, pp. 2270–2275, 2002.

[11] D. Hsu, S. Bridges, M. Figueroa, and C. Diorio, "Adaptive quantization and density estimation in silicon," in *Advances in Neural Information Processing Systems (NIPS02)*, vol. 15, (Vancouver, Canada), Dec 2003.

[12] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science* (B. R. Gaines, ed.), vol. 2, (New York), pp. 37–172, Plenum, 1969.

[13] D. E. V. D. Bout and T. K. MillerIII, "A digital architecture employing stochasticism for the simulation of hopfield neural nets," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 5, pp. 732–738, 1989.

[14] T. G. Clarkson, D. Gorse, J. G. Taylor, and C. K. Ng, "Learning probabilistic ram nets using vlsi structures," *IEEE Transactions on Computers*, vol. 41, no. 12, pp. 1552–1561, 1992.

[15] T. G. Clarkson, Y. Guan, J. G. Taylor, and D. Gorse, "Generalization in probabilistic ram nets," *IEEE Transactions on Neural Networks*, vol. 4, no. 2, pp. 360–363, 1993.

[16] T. G. Clarkson, C. K. Ng, and Y. Guan, "The pram : An adaptive vlsi chip," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 408–412, 1993.

[17] B. D. Brown and H. C. Card, "Stochastic neural compuation i : Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.

[18] B. D. Brown and H. C. Card, "Stochastic neural computation ii: Soft competitive learning," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 906–920, 2001.

[19] J. Alspector, A. Jayakumar, and S. Luma, "Experimental evaluation of learning in a neural microsystem," in *Advances in Neural Information Processing Systems (NIPS91)*, vol. 4, pp. 871–878, Dec 1992.

[20] P. Y. Ting and R. A. Iltis, "Diffusion network architectures for implementation of gibbs samplers with applications to assignment problems," *IEEE Transactions on Neural Networks*, vol. 5, no. 4, pp. 622–638, 1994.

[21] J. Alspector, J. W. Gannett, S. Haber, M. B. Parker, and R. Chu, "A vlsi-efficient technique for generating multiple uncorrelated noise sources and its application to stochastic neural networks," *IEEE Trans. Circuits and Systems*, vol. 38, no. 1, pp. 109–123, 1991.

[22] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 195–281, 1986.

[23] G. E. Hinton, "Products of experts," in *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99)*, (Edinburgh, Scotland), pp. 1–6, Sept. 1999.

[24] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[25] A. F. Murray, "Novelty detection using products of simple experts-a potential architecture for embedded systems," *Neural Networks*, vol. 14, no. 9, pp. 1257–1264, 2001.

[26] P. Fleury, R. J. Woodburn, and A. F. Murray, "Matching analogue hardware with applications using the products of experts algorithm," *Proc. IEEE European Symp. on Artificial Neural Networks*, pp. 63–67, 2001.

[27] J. R. Movellan, "A learning theorem for networks at detailed stochastic equilibrium," *Neural Computation*, vol. 10, no. 5, pp. 1157–1178, 1998.

[28] J. R. Movellan, P. Mineiro, and R. J. Williams, "A Monte-Carlo EM approach for partially observable diffusion processes: Theory and applications to neural networks," *Neural Computation*, vol. 14, no. 7, pp. 1507–1544, 2002.

[29] G. Cauwenberghs, "An analog vlsi recurrent neural network learning a continuous-time trajectory," *IEEE Transactions on Neural Network*, vol. 7, no. 2, pp. 346–361, 1996.

[30] L. O. Chua, T. Roska, T. Kozek, and A. Zarandy, "Cnn universal chips crank up the computing power," *IEEE Circuits and Devices Magazine*, vol. 12, pp. 18–28, July 1996.

[31] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the U.S.A.*, vol. 79, pp. 2554–2558, 1982.

[32] S. Haykin, ed., *Recurrent Networks Rooted in Statistical Physics*, ch. 8, pp. 285–351. In Haykin [34], 2 ed., 1994.

[33] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences of the U.S.A.*, vol. 81, no. 10, pp. 3088–3092, 1984.

[34] S. Haykin, ed., *Neural Networks*. New York: Macmillan Publishing Co., 2 ed., 1994.

[35] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in boltzmann machine," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. Rumelhart, J. McClelland, and the PDP Research Group, eds.), (Cambridge, Massachusetts), pp. 283–317, MIT, 1986.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[37] C. Peterson and J. R. Anderson, "A mean field theory learning algorithm for neural networks," *Complex Systems*, vol. 1, pp. 995–1019, 1987.

[38] J. Alspector, B. Gupta, and R. B. Allen, "Performance of a stochastic learning microchip," in *Advances in Neural Information Processing Systems (NIPS88)*, vol. 1, pp. 748–760, Dec 1989.

[39] A. Jayakumar and J. Alspector, "A cascadable neural network chip set with on-chip learning using noise and gain anealing," in *Proceedings of IEEE Custom Integrated Circuits Conference (1992)*, pp. 19.5.1–19.5.4, 1992.

[40] J. R. Movellan, "A local algorithm to learn trajectories with stochastic neural network," *Advances in Neural Information Processing Systems (NIPS93)*, vol. 6, pp. 83–87, 1994.

[41] J. R. Movellan and J. L. McClelland, "Learning continuous probability distributions with symmetric diffusion network," *Cognitive Science*, vol. 17, pp. 463–496, 1993.

[42] Y. Freund and D. Haussler, "Unsupervised learning of distributions on binary vectors using two layer networks," *Advances in Neural Information Processing Systems(NIPS91)*, vol. 4, 1992.

[43] M. Welling and G. Hinton, "A new learning algorithm for mean field boltzmann machines," in *Proceedings of the Twelfth International Conference on Artificial Neural Networks (ICANN2002)*, pp. 351–357, Aug 2002.

[44] G. Mayraz and G. E. Hinton, "Recognizing hand-written digits using hierarchical products of experts," *Advances in Neural Information Processing Systems(NIPS00)*, vol. 13, pp. 953–959, 2001.

[45] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1257–1272, 1988.

[46] L. O. Chua and L. Yang, "Cellular neural networks: Applications," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1273–1290, 1988.

[47] L. O. Chua and T. Roska, "The cnn paradigm," *IEEE Transactions on Circuits and Systems-I: Foundamental Theory and Applications*, vol. 40, no. 3, pp. 147–156, 1993.

[48] T. Roska and L. O. Chua, "The cnn universal machine: An analogic array computer," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, no. 3, pp. 163–173, 1993.

[49] J. M. Cruz and L. O. Chua, "Design of high-speed, high-density cnns in cmos technology," *International Journal of Circuit Theory and Applications*, vol. 20, pp. 555–572, 1992.

[50] J. Alspector, R. B. Allen, A. Jayakumar, T. Zeppenfeld, and R. Meir, "Relaxation networks for large supervised learning problems," in *Advances in Neural Information Processing Systems(NIPS90)*, vol. 3, pp. 1015–1021, Dec 1991.

[51] P. Fleury, M. Reekie, and A. F. Murray, "High-accuracy mixed-signal vlsi for weight modification in contrastive divergence learning," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN'2002)*, (Madrid, Spain), pp. 426–431, Aug 2002.

[52] P. Fleury and A. F. Murray, "Mixed-signal vlsi implementation of the products of experts' contrastive divergence learning scheme," in *Proceedings of the IEEE International Symposium on Circuits And Systems (ISCAS'2003)*, vol. 5, (Bangkok, Thailand), pp. 653–656, May 2003.

[53] P. Fleury, H. Chen, and A. F. Murray, "On-chip contrastive divergence learning in analogue vlsi," in *Proceedings of the International Joint Conference on Neural Networks*, (Budapest), Jul. 2004.

[54] J. Alspector, A. Jayakumar, and B. Ngo, "An electronic parallel neural cam for decoding," in *Neural Networks for Signal Processing [1992] II., Proceedings of the 1992 IEEE-SP Workshop*, pp. 581–587, 1992.

[55] A. S. Sedra and K. C. Smith, eds., *Microelectronic Circuits*. New York: Saunder College Publishing, 3rd ed., 1991.

[56] J. Choi, S. H. Bang, and B. J. Sheu, "A programmable analog vlsi neural network processor for communication receivers," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 484–495, 1993.

[57] M. Banu and Y. Tsividis, "Floating voltage-controlled resistors in cmos technology," *Electronics Letters*, vol. 18, pp. 678–679, 1982.

[58] P. Moerland and E. Fiesler, "Neural network adaptation to hardware implementations," in *Handbook of Neural Computation* (E. Fiesler and R. Beale, eds.), (New York), pp. 1–13, Institute of Physics and Oxford University, 1997.

[59] C. Mead, ed., *Analog VLSI and Neural Systems*. Addison-Wesley Pub. Comp., 1st ed., 1989.

[60] G. Wegmann and E. A. Vitozz, "Basic principles of accurate dynamic current mirrors," *IEE Proceedings on Circuits, Devices And Systems*, vol. 137, pp. 95–100, 1990.

[61] R. Genov and G. Cauwenberghs, "Stochastic mixed-signal vlsi architecture for high-dimensional kernel machines(nips01)," *Advances in Neural Information Processing Systems(NIPS01)*, vol. 14, pp. 1099–1105, 2002.

[62] R. Genov and G. Cauwenberghs, "Kerneltron: support vector machine in silicon," *IEEE Transactions on Neural Networks*, vol. 14, no. 8, pp. 1426–1434, 2003.

[63] R. Karakiewicz, A. Olyaei, R. Genov, S. Chakrabartty, and G. Cauwenberghs, "Silicon support vector machine," *Advances in Neural Information Processing Systems(NIPS03)*, vol. 16, p. In Press, 2004.

[64] Y. W. Teh and G. E. Hinton, "Rate-coded restricted boltzmann machine for face recognition(nips00)," *Advances in Neural Information Processing Systems(NIPS00)*, vol. 13, pp. 908–914, 2001.

[65] R. J. Woodburn, A. A. Astaras, R. W. Dalzell, A. F. Murray, and D. K. McNeill, "Computing with Uncertainty in Probabilistic Neural Networks on Silicon," in *Proc. 2nd Int. ICSC Symp. on Neural Computation*, pp. 470–476, 1999.

[66] S. ROSS, ed., *A First Course in Probability*. New York: Macmillan Publishing Co., 4 ed., 1994.

[67] B. J. Frey, "Continuous sigmoidal belief networks trained using slice sampling," *Advances in Neural Information Processing Systems(NIPS96)*, vol. 9, pp. 452–458, 1997.

[68] H. Chen and A. F. Murray, "A continuous restricted boltzmann machine with an implementable training algorithm," *IEE Proceedings of Vision, Image and Signal Processing*, vol. 150, no. 3, pp. 153–158, 2003.

[69] L. Tarassenko and G. Clifford, "Detection of ectopic beats in the electrocardiogram using an auto-associative neural network," *Neural Processing Letters*, vol. 14, no. 1, pp. 15–25, 2001.

[70] Y. Wang, Y. S. Zhu, N. V. Thakor, and Y. H. Xu, "A short-time multifractal approach for arrhythmia detection based on fuzzy neural network," *IEEE Trans. on Biomedical Engineering*, vol. 48, pp. 989–995, Sept. 2001.

[71] T. Tang, H. Chen, and A. Murray, "Adaptive Stochastic Classifier for Noisy pH-ISFET Measurements," in *Proceedings of Thirteenth International Conference on Artificial Neural Networks (ICANN2003)*, (Istanbul, Turkey), pp. 638–645, Jun. 2003.

[72] T. Tang, H. Chen, and A. Murray, "Adaptive, integrated sensor processing to compensate for drift and uncertainty: A stochastic neural approach," *IEE Proceeding on Nanobiotechnology*, vol. 151, no. 1, pp. 28–34, 2004.

[73] B. Gilbert, "A precise four-quadrant multiplier with subnanosecond response," *IEEE Journal of Solid-State Circuits*, vol. SC-3, pp. 365–373, 1968.

[74] D. McNeil, C. Schneider, and H. Card, "Analog cmos neural networks based on gilbert multipliers with in-circuit learning," in *IEEE Proceedings of the 36th Midwest Symposium on Circuits and Systems*, vol. 2, pp. 1271–1274, Aug 1993.

[75] D. Coue and G. Wilson, "A four-quadrant subthreshold mode multiplier for analog neural-network applications," *IEEE Transactions on Neural Networks*, vol. 7, no. 5, pp. 1212–1219, 1996.

[76] N. Saxena and J. Clark, "A four-quadrant cmos analog multiplier for analog neural networks," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 6, pp. 746–749, 1994.

[77] G. Colli and F. Montecchi, "Low voltage low power cmos four-quadrant analog multiplier for neural network applications," in *IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 496–499, May 1996.

[78] S. Vlassis and S. Siskos, "Analog cmos four-quadrant multiplier and divider," in *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, vol. 5, pp. 383–386, June 1999.

[79] C. H. Lin and M. Ismail, "A 1.8v low-power cmos high-speed four quadrant multiplier with rail-to-rail differential input," in *IEEE International Conference on Electronics, Circuits and Systems*, vol. 1, pp. 37–40, Sept 1998.

[80] A. M. Ismail and A. M. Soliman, "A novel cmos four-quadrant multiplier based on linearization of the long tail differential pair," in *IEEE International Symposium on Circuits and Systems (ISCAS2000)*, (Geneva, Switzerland), pp. 485–488, May 2000.

[81] S. Liu and Y. Hwang, "Cmos squarer and four-quadrant multiplier," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 7, pp. 119–122, 1995.

[82] A. Pesavento and C. Koch, "A wide-range four quardrant multiplier in subthreshold cmos," in *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 240–243, May 1999.

[83] H. Chible, "Analog circuit for synapse neural networks vlsi implementation," in *The 7th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS 2000)*, vol. 2, pp. 1004–1007, 2000.

[84] R. Geiger, P. Allen, and N. Strader, eds., *VLSI Design Techniques for Analog and Digital Circuits*. New York: McGraw-Hill Publishing Company, 1st ed., 1990.

[85] L. Chen and B. Shi, "Cmos pwm vlsi implementation of neural network," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, vol. 3, pp. 485–488, July 2000.

[86] C. Lu, B. Shi, and L. Chen, "Analogue circuit realization of a programmable sigmoidal function and its derivative for on-chip bp learning," in *The 2000 IEEE Asia-Pacific Conference on Circuits and Systems*, vol. 3, pp. 626–629, Dec. 2000.

[87] A. Annema, "Hardware realisation of a neuron transfer function and its derivative," *Electronics Letters*, vol. 30, no. 7, pp. 576–577, 1994.

[88] M. Conti, P. Crippa, G. Guaitini, S. Orcioni, and C. Turchetti, "A current-driven, programmable gain differential pair using mos translinear circuits," in *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 543–546, Jun. 1998.

[89] E. Vittoz, "Mos transistors operated in the lateral bipolar mode and their application in cmos technology," *IEEE Journal of Solid-State Circuits*, vol. SC-18, no. 3, pp. 273–279, 1983.

[90] J. Lansner and T. Lehmann, "An analog cmos chip set for neural networks with arbitrary topologies," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 441–444, 1993.

[91] G. Cauwenberghs and A. Yariv, "Multivalued mos memory for variable-synapse neural networks," *Electronics Letters*, vol. 25, no. 10, pp. 669–670, 1989.

[92] G. Cauwenberghs and A. Yariv, "Fault-tolerant dynamic multilevel storage in analog vlsi," *IEEE Transactions on Circuits and Systems II: Analog and Digital Processing*, vol. 41, no. 12, pp. 827–829, 1994.

[93] G. Wegmann and E. A. Vitozz, "Very accurate dynamic current mirrors," *Electronic Letters*, vol. 25, no. 10, pp. 644–646, 1989.